(1)

20001013178

A DISTRIBUTED OBJECT-ORIENTED DATABASE APPLICATION DESIGN

THESIS

Hsin-feng (Edward) Wu
Captain, ROCAF

AFIT/GCS/ENG/93M-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/93M-06

A DISTRIBUTED OBJECT-ORIENTED DATABASE APPLICATION DESIGN

THESIS
Hsin-feng (Edward) Wu
Captain, ROCAF

AFIT/GCS/ENG/93M-06

Approved for public release; distribution unlimited

AFIT/GCS/ENG/93M-06

A DISTRIBUTED OBJECT-ORIENTED DATABASE APPLICATION DESIGN

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Hsin-feng (Edward) Wu , B.S.
Captain, ROCAF

March 1993

Approved for public release; distribution unlimited

Preface

The purpose of this thesis effort is to develop a distributed object-oriented database application to support the design and development for parallel software development of the Parallel Algorithms and Applicaticns Group (PAAG) at AFIT.

In developing the application, I received invaluable assistance from many individuals. First of all, I'd like to express my gratitude to Professor Gary B. Lamont, my thesis advisor, for his constant guidance and inspiration throughout this research.

I also thank my thesis committee members, Professor Thomas C. Hartrum and Major Mark A. Roth, for their contributions to this thesis. Without their supports, this work would have not been possible. In addition, I thank Professor Bruce Suter, Dan A. Zambon, Dave Doak, Zheng-Jun Guo and al' the professors, faculties that have helped me either directly or indirectly to accomplish my study at AFIT.

My deepest appreciation goes to my dear wife, Hsien-shan (Angela), whose love, devotion, and morale support kept me going throughout this research and my study at AFIT.

Finally, I want to let my daughter, Meng-shan (Angelina), and son, Meng-han (Edwin) know the importance of knowledge, and I will build them up to understand that knowledge is the source of power.

Hsin-feng (Edward) Wu

Table of Contents

Page

List of Figures

Figure Page

List of Tables

AFIT/GCS/ENG/93M-06

Abstract

The purpose of this study is to analyze and develop a distributed object-oriented database management system(DOODBMS) application to support parallel software development of the Parallel Algorithms and Applications Group (PAAG) at AFIT.

By following the software lifecycle of object-oriented paradigm, this thesis investigation is intended to generate requirements analysis, design, implementation, user interface design and implementation, and integrating test of the database application and user interface. ITASCA, an existing DOODBMS platform, has been chosen to prototype the application because of its distribution functionalities and object-oriented modeling power.

The user interface of the database application is based on X-window OSF/Motif front-end system for its efficiency. Various operations of this distributed database include: retrieve information in the database, modify data at the local site, look up information of publications of libraries, contact support groups for meeting or research conference, schedule meetings for conference or workshop, send meeting reminders to meeting attendees of faculty, student and support personnel, etc. The user may access each local computer through graphical user interface without knowing the query language constructs of the DOODBMS platform.

A DISTRIBUTED OBJECT-ORIENTED DATABASE APPLICATION DESIGN

I. Introduction

1.1 Barkgıound

The technology of distributed database manıgement systems (DDBMS) is the union of database management systems (DBMSs) and communication networks. One major motivation behind the use of DBMS is to integrate the operational data of an enterprise and centralize controlled access to that data(121). On the other hand, the technology of computer networks promote distributed environments of data processing. At first glance, these two contrasting approaches cannot be combined tɔ produce a more powerful technology. The key is that the most important objective of the database technology is integration, not centralization(121). The computer network technology can be applied to DBMS integration without centraiization. This is the objective of the DDBMS technology.

DBMSs have proven themselves very successful in a large variety of computer applications. Today, many effective DBMS platforms are provided for managing large repositories of data while providing access and control to multiple users and applications in a distributed environment. Despite these advantages, there are numerous computer applications that continue to spurn the use of a DBMS in favor of their own unique application file system. Because conventional databases are unable to adequately support these applications, the need for database support becomes evident as the systems proliferate among more powerful workstations and in increasingly complex engineering environments. Distributed Object-oriented database management systems (DOODBMS) have shown the potential for providing the necessary database support for these complex, data intensive applications in distributed environments.

E. F. Codd, the founder of the relational model, asserts that distributed technology is only feasible when based on a relational foundation(90) and Date(38) suggests that the object-oriented database technology is not yet mature enough to replace the relational DBMS. Despite these suggestions, this thesis investigation attempts to examine the potential of distributed object-oriented database applications to support complex design applications of parallel software development for the Parallel Algorithms and Applications Group (PAAG) at AFIT by using a commercially available DOODBMS platform, ITASCA Object Management System.

1.2 Problem Statement/Requirements

The objective of this study is to evaluate the design and create a prototype of a distributed object-oriented application for supporting the design and development of the application on parallel software development environment at AFIT. This distributed DBMS application maintains related information for the following people and organizations: Faculty, Students, Special Interest Groups, Computer support personnel, Libraries, Meetings, Classes, Offering, Thesis, Advisor, Institution, E-Mail, Research, Publications, Job-opportunity, etc..

This application consists of a number of workstations that are interconnected by a high bandwidth local-area communication network. The structure of such a network is highly dynamic. At any time, a new workstation may be added or an existing workstation may be removed from the network. Users of the distributed application are able to access information in the global database. The operations of this distributed database include: retrieve information in the database; modify data at the local site; look up information of publications of libraries; keep track of related thesis work; locate software; electronic copy of thesis, etc.; report forms; contact support groups for meetings or research conference; schedule meetings for conference or workshop; send meeting reminders to meeting attendees of faculty, student and support personnel, etc. The users of this application are mainly from the Parallel Algorithms and Applications Group (PAAG). Assume that users from the group are not equipped with DBMS system-dependent knowledge (such as SQL-like languages), a graphical user interface is, therefore, under considerations.

1.3 Objectives

This thesis effort addresses and researches the following topics:

- Search for and develop a design guideline for DOODBMS application design.
- Explore Object-oriented (OO) technology.
- Explore the Distributed DBMS.
- Survey various OODBMS platforms.
- Survey various OO modeling methods.
- Survey various DDBMS platforms.
- Survey various Object-oriented programming languages (OOPL).
- Evaluate Coad and Yourdon object-oriented analysis (OOA) and object-oriented design (OOD) methodology. (Does Coad and Yourdon's OOA and OOD methodology adequately support the design of Object-oriented systems?)
- Evaluate OOATool[1].
- Evaluate a design of a DOODBMS application using ITASCA System.
- Evaluate the ITASCA DODBMS(65).
- Evaluate the ITASCA Dynamic Schema Editor(68).
- Evaluate the ITASCA DBA Tool(69).
- Modify and incorporate Xnetlib[2] into the OSF/Motif user interface.
- Evaluate X Window user interface design using the OSF/Motif for the thesis application.
- Hide query language constructs into a graphical-representation user interface.

[1] The OOATool is an CASE tool for supporting object-oriented analysis. It was developed by Coad and Yourdon and marketed by Object International Inc.

[2] Xnetlib is developed by Oak Ridge National Laboratory in the late 1992. This software provides an X Window user interface (using Athena widgets) and allow users remote access to the Laboratory to retrieve free softwares and related data.

1.4 Assumptions/Constraints

- Database and Network: The nature of distributed environment is complex. Assumptions are required to simplify and specify the distributed environment:

    1. Assume homogeneous DBMS in the distributed DBMS environments.
    2. Assume local disk capacity is sufficient to hold all the allocated data.
    3. Assume the computer network to each node is completely connected.
    4. Assume processing capability is the same in each node.
    5. Assume no queuing delay in network communications.

    Figure 1.1(121) list the workstation configuration for the thesis application.

- Software and Environments:

    1. All development efforts are conducted on a Sun Sparc II workstations.
    2. This thesis effort utilizes the following software platforms:

        (a) The ITASCA Object Management System (version 2.1.1).

        (b) The Dynamic Scheme Editor of ITASCA.

        (c) The DBA Tool of ITASCA.

        (d) The OOATool from Object International Inc..

        (e) The C++ Preprocessor from CenterLine.

        (f) The X Window System OSF/Motif (Version 1.1 or higher) and Motif Toolkits from QUEST Windows Corporation[3].

1.5 Approach

The general approach for this Application Database design is based on the following steps:

1. Understand and analyze the requirements of the application.

[3] Quest Commercial Motif is Released by Questwin Inc. in November, 1991.

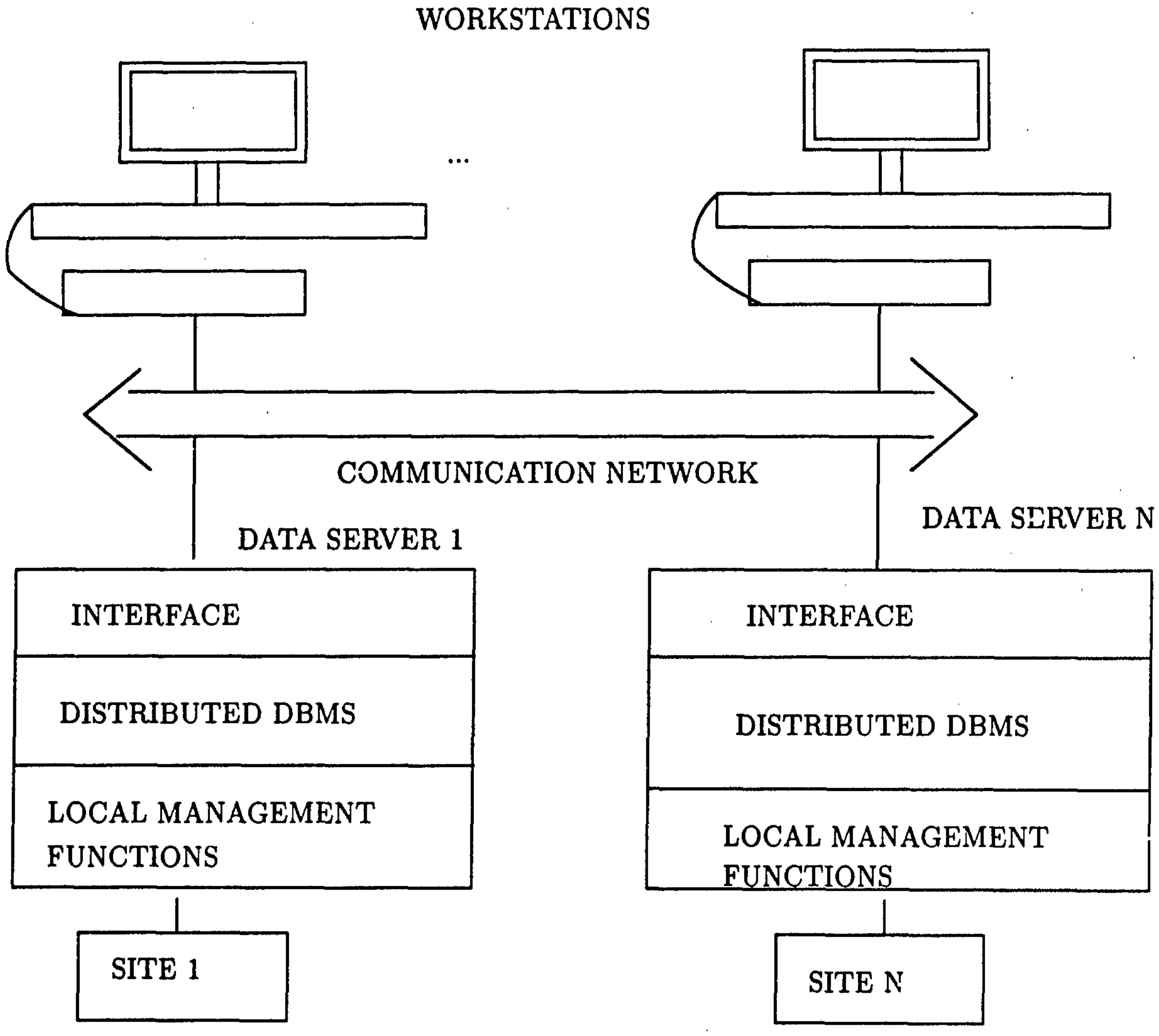


Figure 1.1. Workstations Configuration for the Database Application

2. Develop a high-level DOODBMS application.
3. Consider various DBMSs and select an appropriate one to serve as an platform.
4. Develop a detailed-level distributed object-oriented DBMS design.
5. Design an user interface.
6. Implementation (stepwise fashion).
7. Test, integrate and evaluate.

1.6 Thesis Overview

This thesis is divided into six chapters. Chapter 1 has stated the requirements, summary of current technology, objectives, assumptions/constraints, and approach. Chapter 2 is a literature research which is the fundamental knowledge for developing the application. This chapter also provides a justification for the need of a DOODBMS application with an X window user interface. Chapter 3 summarizes the methodology that has been applied to develop the application. Chapter 4 exemplifies the methodology in analysis and design of the application. Chapter 5 discusses the detailed design and implementation of DBMS and user interface. Chapter 6 presents a conclusion of what has been achieved.

II. Literature Review

2.1 Introduction

During the 1970s, centralized databases dominated the DBMS industry. However in the 1980s and the 90s, with the decentralization of the corporate world and the advances in networking tecbnology, the focus changed to decentralized or distributed databases(139). The problems of centralized DBMSs had become well-understood, and many database systems, including relational database systems, had become commercially available. These systems provided great convenience for information management. Since organizational structures of modern society have changed, people began to realize the impact of some problems caused by having a single, centralized database for an entire organization. For organizations with geographically dispersed units and multiple computers connected by telecommunication lines, accessing a central computer to fetch data has resulted in very low performance and very high cost of communication. Besides, existing commercial DBMS have proven inadequate for complicated applications such as Computer-Aided Software Engineering (CASE), Office Information Systems (OISs), scientific and medical applications, etc.(28), (64), (55), (159), (5), (10). There comes a need for DOODBMS.

The purpose of this chapter is to provide the background knowledge required to understand this thesis investigation and the associated notations used in this thesis. This chapter also presents a rationale for the need of a DOODBMS application using an X window user interface. The basic knowledge of a DOODBMS application design is divided into two major categories: DDBMS and object-oriented design. The approach and methodology to the analysis and design of the distributed database, requirement analysis methodologies are applied to these two categories. Also, a brief functional summary of ITASCA is presented.

2.2 DBMS Definitions

The thesis also presents an explanation of the terms that have been used. The terminology were divided into three categories: Object-oriented technology, DDBMS, and computer network. As we noticed that there is no standard terminology in object-oriented paradigm for the research community. Like the various object-oriented programming languages, terminology differs among methodologies. A brief definition of the terminology of DDBMS and Computer network were also provided for the better understanding of this thesis. Appendix A lists the meanings of the terms that have been used in these three areas.

DDBMS is divided into two major categories: DBMS and computer network. The following definitions provide a background knowledge for these two technologies:

2.2.1 Database Management System (DBMS). A DBMS is composed of a collection of data and a set of programs to access that data. The data in a DBMS is organized according to some data models, such as functional, relational, hierarchical, or object-oriented models. Users are then usually able to access the data through an interface (e.g., the query language) provided by the DBMS. The primary goal of a DBMS is to provide an environment that is both *convenient* and *efficient* to use in retrieving and storing database information(89).

2.2.1.1 Traditional DBMS. A conventional DBMS commonly has the following capabilities or functions(89), (38), (5), (76):

1. A data model: this model provides a set of structures that are used to model the information.
2. Transaction management: a data manager that monitors database transactions to ensure data consistency and stability.
3. Large, persistent record storage: the data of a DBMS must be persistent between application programs execution.
4. Query language: a high-level language for accessing data, such as SQL.
5. Data independence: the ability to be able to change the data stored in the database without modifying existing application programs. For instance, in an object-oriented model, physical data independence is achieved by using a declarative query language in programs and methods.

6. Concurrency: multiple users access the same database simultaneously, and may interact with each other.
7. Crash recovery: a DBMS must have the ability to recover from a crash.
8. Change Management: database support for managing the change of data throughout the database life cycle.
9. Security: protect the data from unauthorized access.

2.2.1.2 Object-oriented DBMS. An Object-oriented DBMS (OODBMS) is a DBMS that the data is organized according to an object-oriented model and the design is based on the object-oriented design. This design methodology embraces the features of Object-oriented Programming (OOP), such as Abstract Data Type (ADT), encapsulation and inheritance, etc. With capabilities of traditional DBMS, an OODBMS should support the following additional capabilities(28), (64), (5), (76):

1. Data abstraction: this allows the development and use of abstract and logically complex and flexible types.
2. Powerful information modeling capabilities: information is modeled in the form of classes and objects that represent and capture the structures and behaviors of real world entities.
3. Unique object identifiers: this ability to generate identifiers for objects represented in the database.
4. Composite objects: most design applications require the ability to define objects that contain other objects, such that all the subcomponents and their subcomponents act as a single object for the purposes of database operations.
5. Encapsulation and data-hiding: in object-oriented DBMS, objects are manipulated by operations that are defined on their types, so that data-hiding can be achieved.
6. Inheritance: this allows the reuse of new class structures in terms of existing ones;
7. Ease of schema changes: it is important to be able to modify a schema with minimum impact on existing applications. This feature is even more important in design

application, because the users as well as the application programmers may modify the schema.

8. Seamlessness: integration of the database with the rest of the programming environment in a non-obtrusive manner.
9. Message passing: the interaction of objects by the innovation of each others' methods.
10. Polymorphic data and functions: this is a feature in which data dynamically assumes various forms determined at routine.
11. Extensibility: new operations and types can be incrementally added to the application.

The users of object-oriented database technology generally are from two distinct worlds: Object programmers and Database application developers. The former group is largely C programmers who have moved to C++. They expect OODBMSs to provide database functionality with their environments. The latter is using 4GLs and SQL-like languages to support the performance and distribution needs of emerging applications with complex data and graphical user interfaces. Database application developers can be more productive with object technology if they have tools that leverage their 4GL and SQL expertise(99).

Current researches appear to be progressing in two directions represented by the "Third Generation Data Base System Manifesto"(159) and "The Object-Oriented Database System Manifesto"(10). The former group seems to be progressing toward enhanced relational database research, by integrating object-oriented principles into current relational database technology. This approach combines the maturity and robustness of the DBMS with the performance of the object-oriented programming languages(133). OpenODB(4) adopts this approach. The latter group excels at writing system-level code and mastering the OOPL, such as C++. They focus on writing class libraries and extend object-oriented programming languages. ObjectStore(92), ORION (ITASCA)(82), GemStone(102), ONTOS DB(56), VERSANT ODBS(56) and Objectivity DB(56) illustrate this approach. Cattell(28) suggests that both directions will continue, with the enhanced relational research aimed toward business applications, while the latter research targeted the scientific and object-based applications(28).

Yet another approach is implementing an OODBMS by extending an relational DBMS (RDBMS) with new data types, operators, and access methods. This type of OODBMS integrates well with existing RDBMS and provides for smooth data flow between engineering and business applications. Potential disadvantages are performance and robustness limitations. Even an augmented RDBMS may not be capable of efficient operations on individual objects(133). POSTGRES(158) and EXODUS(27) are two examples.

In "The Object-oriented Database System Manifesto"(10), thirteen mandatory features and five optional features of OODBMS were defined:

- Mandatory Object Features:
    1. Support complex objects.
    2. Support object identity.
    3. Object encapsulation.
    4. Object types or classes.
    5. Class or type inheritance.
    6. Late binding with overriding and overloading.
    7. Computational completeness.
    8. Ability to add new types (extensibility).
- Mandatory Database Features:
    1. Persistence of objects.
    2. Ability to manage very large databases.
    3. Concurrent user support.
    4. Recovery from hardware and software failure.
    5. Ad hoc query support.
- Optional Features:

1. multiple inheritance.
2. type checking and inferencing.
3. distributed system.
4. long transactions.
5. versioning.

2.2.1.3 Distributed DBMS (DDBMS). A DDBMS consists of a single logical database that is distributed physically either over a geographical area (such as the United States) or over a small geographical area (such as a single building or a number of adjacent buildings)(89). In other words, a distributed database is composed of a collection of sites, each of which represents one computer and its secondary storage devices. There are two characteristics of a distributed database: 1) the environment is a computer network, 2) data is distributed to the nodes of the network by duplicating and partitioning files.

A DDBMS also has a single DBMS that provides consistent queries and updates. Put simply, a DDBMS requires the integration of a communication network and implies homogeneity between all the physical components. That is, it supports just one data model and query language, with an unambiguous schema. To the users, distributed system should look like a non-distributed system and the problems of distributed functions are internal to the system not external to the users.

The definition of a DDBMS varies among theorists, vendors, and users. But, DDBMS should not be confused with distributed processing, where the processing rather than the data is distributed (although DDBMSs allow distributed processing). A true DDBMS does not allow just remote data or database access, network file system support, or the ability to access data at multiple sites at the same time. Date(38) suggested twelve rules for establishing a DDBMS:

1. Local autonomy: local data is managed independently of other sites, i. e. no site X should depend on another site Y for its successful functioning. Local autonomy also implies that local data is locally owned and managed.

2. No reliance on a central site: no DBMS site is any more important than any other. All sites should be treated equally.

3. Continuous operation: no planned activity (such as adding a new site, or upgrading the DBMS) should require a shutdown.

4. Location independence: users and programs do not need to know the location of the data. The users should be able to access data as if the data was stored at their own local site.

5. Fragmentation independence: a table that has been fragmented will appear as a single table to users and programs. Fragmentation is desirable for two reasons: enhance performance and reduce communication costs.

6. Replication independence: redundant data will be transparently updated. Replication is desirable for two reasons: enhance performance and better availability.

7. Distributed query processing: queries should be optimized for the distributed database, especially, when query process involves multiple sites.

8. Distributed transaction management, including recovery and concurrency control is desirable.

9. Hardware independence: different hardware systems all participate as equal in a distributed system.

10. Operating system independence: different operating system either on the same or different hardware all participate as equal in a distributed system.

11. Network independence: different communication network can be supported.

12. DBMS independence: different DBMSs all participate (to some degree) in a distributed (heterogeneous) system.

2.2.2 Communication Network. The combination of computer and communication technologies have created the computer network. A computer network is an interconnected collection of autonomous computers that are capable of exchanging information among them(121). The autonomous computers are called nodes, hosts, or sites of the network and they are interconnected by means of communications links such as telephone lines, coaxial cabies, fiber optics, or satellite links. In a distributed network, the nodes are geographically distributed and the computers at each node communicate over the communication links. Each node is capable of sharing the resources of every other node.

A database spread over several computers of a computer network is called distributed database. A distributed database management system (DDBMS) is defined as the software system that permits the management of the DBMS and makes the distribution transparency to the users(121). As pointed out by Ozsu(121), personal workstations with a power of 3 to 10 MIPS and fast communication networks are becoming common and this trend will accelerate in the near future. Bit-mapped workstations significantly enhance end-user productivity through human interfaces. Therefore, economics will force the replacement of cheap terminals by workstations acting as application servers. The need to integrate a variety of workstations connected by a local network with a data server has resulted in a new and popular organization of computer resources.

Figure 1.1 shows a simple organization of a distributed server. One promising application is to treat each distributed data server as a single site of a DDBMS. In this way, a distributed database whose sites are connected by a communication network, each site has a single data server connected by a local area network to a cluster of workstations. Any workstation could access the data at any data server through a local area network or wide area network(121).

2.3 Why a DDBMS?

Before addressing this issue, let us compare some major DDBMSs.

- Some Major DDBMSs: Table 2.1 lists a comparison of some major DDBMSs. For more detailed information about DDBMSs, refer to (106), (65), (5), (14), (30).

  Why have a DDBMS? Let us consider the benefits and disadvantages as follows:

- Some Benefits of DDBMS: There are several reasons why DDBMSs are developed. The following is a list of the main motivations:

System	D. F.	D. R.	C. C.	Q. P.	C. Q.	D. C.	D. M
DDM	H	Yes	Locking	Centralize	Yes	Detection	Functional
DDTS		Yes	Locking	Semi-Central.	Yes	Prevention	E-R Model
INGRESStar	H	No	Locking	Centralize	No		Relational
ENCOMPASS	H	No	Locking	Centralize	No	Time-out	Relational
POLYPHEME	No	No	None	Centralize	No	No	Relational
POREL	H	Yes	Locking	Centralize	Yes	Avoidance	Relational
R*	No	No	Locking	Semi-Central.	Yes	D. D.	Relational
SDD-1	HV	Yes	Timestamp	Central.	No	Avoidance	Relational
SIRUS-DELTA	HV	Yes	Locking	Centralize	Yes	Prevention	Relational
VDN	H	Yes	Locking	Centralize		Prevention	Relational
Prime	H	Yes	Locking	Centralize			Network
Multibase	Yes	Yes	No	Retrieval-Only	No	Detection	Functional
ORION/ITASCA		Yes	Locking	Distributed	yes	Detection	OO
Objectivity DB		Yes	Locking	Distributed	No	Detection	OO

Legend:
D. F. = Define Fragments.
D. R. = Define Replica
C. C. = Concurrency Control
Q. P. = Query Planning
D. D. = Distributed Detection
C. Q. = Compile Query
D. C. = Deadlocks Control
D. M. = Data Model
H = Horizontal
HV = both Horizontal and Vertical
OO = object-oriented

Table 2.1. Comparisons of Some Major DDBMSs

1. Organizational and Economic Reasons: The local autonomy issue is very important in those organizations that are inherently decentralized. Consider the ability to partition the authorities and responsibilities of information management for a large organization. This is the major reason many business organizations consider distributed information systems(121). For the economic reason, the arrival of minicomputers and microcomputers has revolutionized the usage of computers. It normally costs much less to put together a system of smaller computers with the equivalent power of a single machine(121).

2. Reliability and Availability (R&A): In DDBMS, data replication is used to obtain higher R&A(30). Media failure, machine failure, network partitioning isolation, network failure or a crash of one site does not cause the whole system to become inaccessible. Even though some data may be inaccessible, the DDBMS can still provide limited service(121). Besides, R&A is crucial to the competitive business world. Storing replicated data in more than one site could also serve as a backup of the files that make the whole system robust.

3. Interconnection of Existing Databases: When an organization needs to integrate some already existing DBMSs, DDBMS is a natural solution. Frequently the DBMSs are supplied by different vendors and may support different data models. One common approach is to integrate these databases through a single user interface. The effort required to interconnect these systems is normally much less than that needed for the creation of a completely new centralized database(30).

4. Performance Considerations: Accessing a centralized database from a remote site may involve time consuming processes that slow down the database performance. The increase of performance of DDBMS can be achieved by parallel processing from several local processors and by partitioning data into segments of interest to different classes of users. This leads to achieving a better database performance.

5. Reduced Communication Overhead: The communication network is the bottleneck of many centralized database systems when a large amount of data is

transferred among them. The fact is that DDBMS with local processing capabilities clearly reduces the communication overhead with respect to centralized DBMS(30).

6. Configuration Independence: When a centralized computer system becomes saturated, it must be replaced by a more powerful centralized system, or some application programs must be moved to another system. However, update and conversion may involve many hardware changes, and application programs may have problems accessing data stores on other machines (94). Adapting a DDBMS is a good solution. The problems are minimized by integrating a new computer into a distributed system. Configuration independence enables the enterprise to add or replace hardware without changing the existing software components of the DDBMS(94), (14), (24), (71), (30), (121).

7. Location and Replication Independence: This feature enables front-end users to access data without knowing at what site the data is stored. Also, since the data is stored in different sites, if a record is updated, then the record in different sites must be changed accordingly. This could be a nuisance to users. Thanks to the replication independence that triggers the changes of a DDBMS and hides the existence of data being stored at different sites. The front-end users may feel like there is only one copy at a single site(94).

8. Scalability: distributed systems offer great flexibility in configuring a total system by the following(122):

   (a) Nodes can be added to or removed from the network.

   (b) Nodes can be reconfigured to meet the changing requirements. For example, disk may be moved from one node to another, some data may be moved from one node to another; some data may be replicated at several nodes.

   (c) Each node may be more or less powerful. For example, a node may be replaced by a multiprocessor node to increase its processing power.

9. Data Security: One of the major benefits of centralized DBMS has been the control it provides over the access of the data. Security can be control in the

sense of controlling the data in one location. The data in one central location is a complete copy, once the hacker break-ins, the security problems occur. On the other hand, For DDBMS, it is more difficult to achieve data security in the sense of maintaining adequate security over computer network. However, it is a natural data security solution of splitting information to different fragments to different nodes. Suppose the communication networks have been well-controlled, the DDBMS could achieve a better data security control.

- Some Limitations of DDBMS: As suggested in (121), (14), (30), (24), (71), some limitations deserve attentions:

  1. Lack of Experience: Commercial DDBMSs are still not commonly known. Most of the existing DDBMSs are custom-made. Some major DDBMS are either under prototyping or are tailored to some limited functions of applications. Limited current knowledge scares people away from using it.

  2. Complexity: Many factors make the DDBMS more complicated than centralized DBMS:

     (a) Data may be replicated.

     (b) Data have been distributed over many sites. The hardware, software, and communication links must also be considered.

     (c) Synchronization of transactions on multiple sites is considerably harder than centralized DBMS.

  3. Difficulty of Change: Lack of experience to tackle the complexity of Distributed DBMS makes the designed and implemented system hard to change. Besides, most of the distributed system have been designed in according to hardware and software configuration. Therefore, it is very difficult to change.

     The complexity and difficulty of change of DDBMS can be handled by object-oriented paradigm as the literature research continues explaining in the next section.

2.4 Why an Object-oriented Paradigm?

According to Martin and Odell(103), object-oriented technology is the catalyst of software industrial revolution in the 1990's. Powerful technologies like CASE, Visual Programming, Code Generator, Parallel Machines, Inference Engine, etc. can be tied together by using object-oriented technology because of the uni-conceptual model in analysis, design, and implementation.

In DBMS, there are five types of data models currently exist: hierarchical, network, relational, object-oriented, and knowledge-based. Why an object-oriented data model? There has been some major change in the past decade. One major changes is the widespread acceptance of relational DBMS. However, existing commercial RDBMS have proven inadequate for complicated applications (such as Computer-aided Software Engineering (CASE), Office Information Systems and scientific and medical applications, etc.) An OODBMS is a DBMS aimed at satisfying the needs of the complicated applications that were mentioned above. Object-oriented paradigm has three major parts: object-oriented analysis, object-oriented design, and object-oriented programming. ODBMS is based on Object-oriented database design. Before discussing the paradigm, let us compare some major OODBMSs.

- Some Major OODBMSs: Table 2.2 lists a comparison of some major OODBMSs.

  For more information, refer to (159), (158), (4), (28), (65), (73), (80). For a commercial information and detailed comments of these systems, refer to (56).

- Advantages of OODBMS over RDBMS for Engineering Applications: DBMSs have proven themselves in a large variety of computer applications. Today's commercial DBMSs provide effective tools for managing large repositories of data while providing access to multiple users and applications. OODBMSs have shown the potential for providing the necessary database support for these complex, data intensive applications because the following capabilities have been added:

  1. A consistent data model: Traditional database development has four conceptual models: analysis, design, programming, and database. For the transition

System	I. L.	M. S.	E. L. I.	M. I.	D. S. F.	L. M.	C. N.
1.	Common lisp, C	text, audio, bit-map image	C/C++, Lisp, Fortran	Yes	Yes	P. L.	Yes
2.	C/C++	No	C/C++	Yes	Limited	Logical	No
3.	Smalltalk	No	Smalltalk, C/C++, GeODE	No	Limited	Logical	No
4.	C/C++	bit-map image	C/C++	Yes	Limited	Logical	No
5.	C/C++		C/C++	Yes	No	Logical	
6.	C/C++	No	C/C++	Yes	Limited	P.L.	Yes
7.	C/C++	No	C/C++	Yes	Limited	Logical	Yes
8.	C/C++	Yes	CO2	Yes	Limited	Logical	
9.	C	multi-media	C/C++, Cobol Fortran, Pascal	Yes	Limited		
10.	C++	No	C/C++	Yes	Limited	Logical	
11.	C	No	C	Yes	Limited	Logical	No
12.	Common Lisp	No	Common Lisp	Yes	Limited	Logical	No

Legend:
1. = ORION/ITASCA (version 2.0)
2. = Object Store (version 1.2)
3. = Gemstone (version 3.0)
4. = ONTOS DB (version 2.2)
5. = Objectivity DB (version 1.2)
6. = ObServer/ENCORE
7. = VERSANT ODBS
8. = O2 (version 3.2)
9. = OpenODB
10. G-BASE (version 4.3)
11. IDB Object Database (version 1.1)
12. Statice
Abbreviation:

I. L. = Implementation Language
M. S. = multimedia Support
E. L. I.= External language interface
M. I. = multiple inheritance
D. S. F. = Dynamic Schema Evolution Facilities
L. M. = Lock Management
P. L. = Physical and Logical Locking
C. N. = Change Notification

Table 2.2. Comparisons of Some Major OODBMSs

between each model, according to Martin(103), there is a "conceptual wall" lies in between. In contrast, Object-oriented technology uses one consistent model. This model combines analysis, design, programming, and OODBMS.

2. A more "realistic", "powerful" data model(5): In RDBMS, there is no direct mechanism to associate behaviors (functions) with data. However, this can be easily achieved with objects in OODBMS. Moreover, Object data representation is more flexible. It allows data to be polymorphic. Multiple inheritance and schema evolution facilities allow the design to grow incrementally.
3. "Easier" schema development: In OODBMS, inheritance allows the schema to be "better" structured to capture the semantics of the application(5).
4. "Better" support for cooperative work: OODBMS allows more complex data handling, and the system is "easier" to extend and maintain(78).

Due to these "powerful" capabilities, much effort has gone into development of object-oriented design practices which allows better modeling of real world objects. The characteristics of the object-oriented paradigm overcome many deficiencies of conventional database systems in solving more complex systems.

The arrival of Object-Oriented Programming (OOP) languages have revolutionized the computer society. This programming paradigm subsumes the concept of abstract data types that define a public and private portion of data structure called object(28). In a value-based data model such as the relational model, data is identified by values. This way of identifying data leads to several problems. One major problem is that the modeling of relationships among data leads to data redundancy (use of foreign keys)(121). Another important problem is the lack of precise semantics for updates(121).

These problems disappear in the object-oriented model. Because this model adopts a "natural" integration with OOP languages that is made possible by object identity and the capability of modeling arbitrarily complex data structures. Boundaries between phases are blurred in the object-oriented software life cycle because objects are the items of interest in all phases. Both the analysis and design phases identify objects and the relationships between objects, providing a continuity that creates a seamless interface between phases. The object-oriented development process is iterative.

- The Motivations and Benefits of Object-Oriented Design: As Coad and Yourdon suggested(34), (35), the Object-oriented technology has the following motivations and benefits:

  1. Bring more understanding of problem domains to analyst.
  2. Improve analyst and problem domain expert interaction.
  3. Increase internal consistency of analysis results.
  4. Tackle more challenging problem domains.
  5. Build system resilient to change.
  6. Reuse OOA, OOD, and OOP results.

Booch (23) suggests the following benefits of applying the object model:

1. Exploits the expressive power of all object-based and object-oriented programming languages.
2. Encourages the reuses of software components.
3. Leads to systems that are more resilient to change.
4. Reduces development risk.
5. Appeals to the working of human cognition.

Martin and Odell(103) suggest the benefits of OOA and OOD as follows:

1. Reusability
2. Stability
3. Reliability
4. Faster design
5. Higher productivity
6. More realistic modeling
7. Easier maintenance
8. Dynamic life cycle
9. Interoperability
10. Massively distributed computing
11. Parallel computing etc..

These three sources agreed that the object-oriented paradigm can solve more complicated problems, build systems resilient to change, and reuse tne software components. Object-oriented prin-iples such as inheritance, abstraction, and encapsulation do enhance the development process by allowing extendibility, simplifying complexity, and increasing reusability.

- The Risks of Object-oriented Design: Booch (23) suggests that therc are two areas of risk which should be consiaered:

  1. Performance Risk: Booch (23) points out the following performance risk:

     (a) For method invocations that cannot be resolved statically, an implementation must do a dynamic lookup in order to find the method defined for the class of the receiving object. Studies show that a method invocation may take from 1.75 to 2.5 times as long as a simple subprogram.

     (b) Another source of performance overhead, according to Booch (23), comes not so much from the nature of the object-oriented programming language but from the way they are used in conjunction with object-oriented design. Because object-oriented design leads to the creation of systems whose components are built in layers of abstraction. Invoking a method at a higher level of abstraction usually results in a cascade of method invocations.

     (c) The encumbrance of classes: a class deep in an inheritance lattice may have many superclasses, whose code must be included when linking in the most specific class. In this way, the task is more complicated, thus increase the performance risk.

     (d) The paging behavior of running applications: Most compilers allocate object code in segments, with the code for complicated unit placed in one or more segments. If the page fault occurs, the performance risk in-reases.

     (e) The dynamic allocation and destruction of objects: Allocating an object on a heap is a dynamic action, however, heap allocation usually costs more computing resources, especially for time-critical applications.

2. Start-up costs: Start-up costs are a big barrier to adopting the object-oriented design. When any organization starts to use object-oriented programming languages for developing applications, they usually have no established base of domain-specific software to reuse. They must start from beginning to figure out how to interface their object-oriented applications with non-object-oriented applications. Therefore, the start-up costs are relatively high.

2.5 Why an X Window System?

The X Window System is an "industry-standard" software system that allows programmers to develop portable graphical user interfaces for applications. This System provides various advantages for application developers. Among them, network transparency, graphical user interface, standard communication protocol, and independence of application design are the major concerns.

1. Network Transparency: The X Window System is a portable software standard developed at the Massachusetts Institute of Technology's Project Athena. This System is a network-transparent window system which allows users to run multiple applications simultaneously in windows, generating text and graphics in monochrome or color on a bitmap display. Network transparency means that we can use application programs that are running on other machines throughout the network, as if they are running on the machines that we use. For example, an application requiring a large shared database can run on the CPU containing that database, and connect to users' workstations over the network using the X Window System(75).

2. Graphical User Interface (GUI): The X Window System is a software environment for engineering workstations. This System offers a rich and complex environment to the users. The central dogma of the X Window System is that the base window system provides mechanism, not policy. For example, the X Window System allows windows on the screen to overlap one another, and allows output to partially obscured windows. User interface design should be a prime concern for the designer of any computer system because it can increase the speed of learning of the system, increase

the performance of the system and users, reduce the error rate, encourage rapid recall of how to use the interface.

3. Standard Communication Protocol: The X Window System is a standard communication protocol that permits applications to be device-independent, applications need not be rewritten, recompiled, or relinked to execute on new display hardware. Although the X Window System is fundamentally defined by a network protocol, the application programmers need not worry about bits, bytes, and message formats when they design applications for workstations, because the X Window System provides interface library for application programmers. A toolkit, such as Widget, consists of an extensible set of predefined graphical components or widgets such as menu, bottom, scroll bars, and text editors, but its usage required an extensive learning of the system and significant program efforts.

4. Independence of Application Design: User interface design is, ideally, independent of application design. Dialogue independence(57) is the feature characteristic of an interactive software system design. In this design, Seeheim model(52) is proposed to separate the design of user interface dialogue from the design of computational applications so that changes in either tend not to cause changes in the other. This independence is achievable using the X Window System because of the user interface could be designed with toolkits provided by the system without any serious concern of the application. Such system design methodology supports rapid prototyping of user interface application. The user interface could be implemented and tested to satisfy the user. Textual-driven or command-driven user interface are usually less independent because of function calls must be implicitly specified in the application programs.

2.6 Current Knowledge of DOODBMS

DOODBMS is one of the challenges of the so called "next-generation" DBMS. During the past two decades, among the various directions that the database research community has explored, relational DBMS (RDBMS). transaction management systems, and distributed relational DBMSs have the most impacts and successes(153). However, DOO-

DBMSs remains a challenge to the research community, partly due to the fact that object-oriented technology has yet to be defined, and partly because of the interoperability with the existing systems has yet to be provided to users.

In the past decade, there was a realization that organizations are fundamentally decentralized and required databases at multiple sites. This requires researchers to investigate new algorithms for distributed query optimization, concurrency control, crash recovery, and support multiple copies of data objects for higher performance and availability. Meanwhile, the idea of object-oriented technology, following the concepts of object-oriented programming, to allow handling more complex information for the next-generation DBMS have been extensively investigated over the last few years. As application developers concern about how to store complex data such as images, audio, graphics and documents etc. to fit the need of their organizations, there are now several commercial object-oriented database systems with built-in distributed database capabilities, such as Objectivity/DB, ObjectStore, ITASCA, and Gemstone.

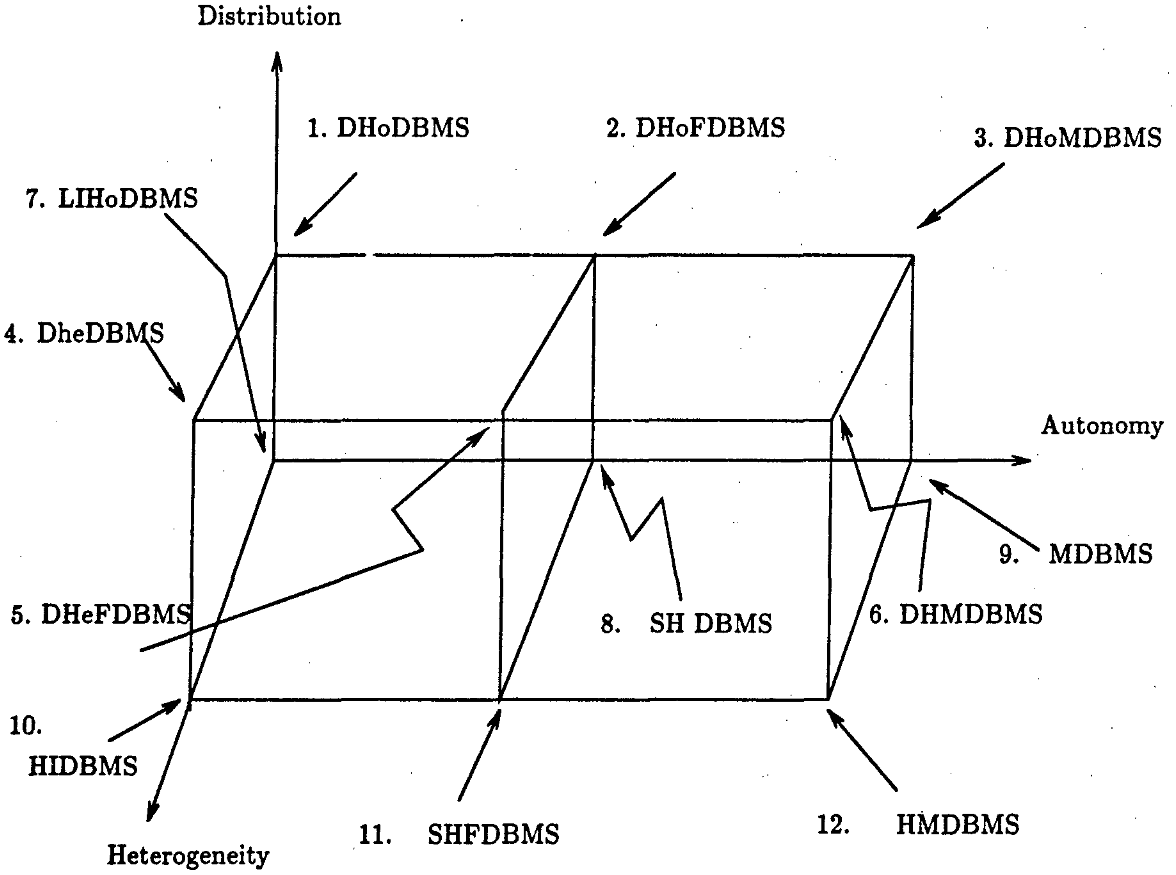
Objectivity/DB and ITASCA are examples of fully distributed object management database. They are fully distributed in both data and control. On the control side, it means you can use multiple databases. Each database is a unit of control, so control can be distributed. Objects can be distributed across multiple databases. Other alternatives to distribute data for OODBMS is to use client/server architectures stored procedures and triggers, such as Sybase. Procedures are typically SQL to access databases. Triggers are stored procedures that execute automatically when a change is made to a table, such as insertion, deletion, or update. In this approach, databases are increasingly being distributed to the various sites. These DDBMSs provide for transparent acquisition of data. However, the local transparency and autonomy is limited because the data is never distributed(123). Client/server architectures are the first step toward true distribution where there is peer-to-peer communication(156). The increasing interest of distributed object-oriented techniques in DBMS is a major source of research within the database community.

Most systems are multiple client/single server that are really not DDBMSs since data is never distributed. Some systems (Oracle Version 6, for example) allow multiple servers, but restrict each client to access only one server at a time. This is distributed only as far as physical organization of the system is concerned (there are many workstations that are interconnected by a computer network) but it is not truly a DDBMS since data is never distributed. Once systems move toward multiple client/multiple server architecture, then we have peer-to-peer communication in DDBMS.

Although DOODBMS is now commercially available, up to this point few books and technical reports addressed in this area, especially, object placement, distribution design, and distributed view. The functions provided in this area are usually system-dependent, for instance, in ITASCA, object placement in distributed environment can be achieved in three ways: 1) create new objects, 2) create new versions, 3) declare shared attributes.

Ozsu et al (122)have devised a classification of design alternatives along three most important factors of DDBMS design considerations: Autonomy, Distribution, and Heterogeneity. Figure 2.1(122) shows the alternative system architectures based on this taxonomy. This thesis only deals with distributed homogeneous object-oriented database systems application design.

According to Ozsu(121), three assumptions have been made to facilitate the progress of distributed databases: 1) assume each site in the computer network is a general purpose computer that executes both application programs and distributed management functions, 2) the computer network is either a wide area network or local area network, and 3) database management is based on the relational data model. However, the advances in hardware technology and software design methodology allow DBMS solve more complex problems. As DDBMS and OODBMS can each solve a subset of problems presented by the new applications, it is more integrated in the future to combine their respective benefits. The most significant impact of computer architecture on distributed databases probably comes from the influence of workstations and parallel computers. By adopting this system architecture, a specialized server can be created, rather than using a general-purpose computer. This relaxes the first assumption. The relaxation of the second assumption leads to another trend— building parallel data servers for distributed databases. And the relaxation of the third assumption leads to other data modeling technique (i. e. object-oriented data model).



Notations:
1. DHoDBMS: Distributed homogeneous DBMS
2. DHoFDBMS: Distributed homogeneous federated DBMS
3. DHoMDbMS: Distributed homogeneous multidatabase system
4. DHeDBMS: Distributed heterogeneous DBMS
5. DHeFDBMS: Distributed heterogeneous federated DBMS
6. DHeMDBMS: Distributed heterogeneous multidatabase system
7. LIHoDBMS: Logical Integrated Homogeneous DBMS
8. SHDBMS: Single-site Homogeneous DBMS
9. MDBMS: Multidatabase systems
10. HIDBMS: Heterogeneous integrated DBMS
11. SHFDBMS: Single-site Heterogeneous DBMS
12. HMDBMS: Heterogeneous Multidatabase systems

Figure 2.1. DDBMS Design and Implementation Alternatives

The need for better distributed technology and object-oriented technology also depends upon CASE tools to make better use of the functionality that is already available. Heterogeneous distributed database is one major concern in the near future, people are not likely to give up their systems for new technology, they rather adopt a new technology to integrate the existing systems. The interoperability, which makes heterogeneous distributed databases behave as if they formed part of a single database, remains a challenge. The problems of incompleteness and inconsistency of existing DBMS which assume system-dependent functions (i.e. transaction management and query) will make the issue more challenging.

2.6.1 Some Expectations of DOODBMS. DOODBMS is in the infant stage. The current attempt to extend relational query languages into object-oriented query languages has results in languages that are not compatible with some object-oriented concepts(84). Many expectations remain to be achieved. Among them are: standard SQL, standard OO design model, DDBMS unsolved problems, etc..

1. Standard SQL: SQL89, a revision of the original SQL standard approved in 1986 (SQL86), is the current standard. This standard assumes the relational DBMS(148). SQL2 standards includes SQL-connection for client/server associations.

   Object-oriented SQL standard, SQL3, which the ANSI committee is also working on, contains triggers, stored procedures, and many features to implement object-oriented capabilities, such as the definition of abstract or user-defined data types, and the definition of a class hierarchy. SQL3 also includes the procedural capability necessary to define stored procedures and methods, or user-defined functions. Hopefully, this will enable the relational model to implement at least some benefits of the object "model" without sacrificing the rigorous mathematical foundation of the relational. This standard is expected to be published in the mid-nineties or later(148).

   Since the standard SQL for object-oriented DBMS is fledgling, vendors are not waiting for standards to appear. Some vendors have released OODBMS products, such as Hewlett-Packard has announced Open Object Database (OpenODB). This product provides a bridge between an object-oriented front-end development environment and an RDBMS on the back end for storing the data. With OpenODB, HP has extended SQL into Object SQL (OSQL). OSQL is a database language for OODBMSs supporting user-defined operations in addition to traditional data definition and manipulation. It combines the advantages of object-oriented programming: abstraction, encapsulation, extensibility, and reuse with advantages of the relational data definition languages (DDL) and data manipulation languages (DML): declarative queries, set-oriented access, views, and authorization(148).

Vendors are incorporating more distributed capabilities in their database products. Users are particularly anxious for vendors to offer features germane to simple client/server architectures such as request options, optimized join, two-phase commit and snapshot capabilities. Users typically request data from one database residing on one server via a remote request or remote transaction. Advanced access methods known as distributed requests and distributed transactions would enable users to send multiple SQL statements that access many servers simultaneously. Optimized join technology reduces the amount of data that gets transmitted from remote databases(46).

2. Standard OO Design Model: There is no single *standard* object-oriented design model. Dozens of well-known OO design methodologies such as OMT(144), Coad and Yourdon(34), (35), Booch (23), Martin and Odell(103), Shlaer and Mellor(150), Wasserman and Pircher(171), Wirfs-Brock et al. (175), etc.. Each model has its own pro and con. For instance, OMT includes an object diagram, data-flow diagrams and state-transition diagram. These diagram, taken together, represent three different, important views of the system. However, they are not well integrated. It is difficult to recognize and implement how an object, behavior or attribute in the object model relates to a data flow or data flow process in the functional model or an event or state in the dynamic model(107), (60), (33), (169). Besides, a key idea of using object-oriented analysis is to obtain a good user interaction and communication with system analysts in the analysis phase(34), (103), (5), OMT embracing three different structure models seems to assume that the user should understand these three models in order to achieve a good user-analyst communication. Certain methodologists

used them for political reasons: facilitating the transfer to the OO paradigm(42). No dominate model in this point. Some models are provided with automated tools (such as OMT, Coad and Yourdon's OOATool and Object Tool). Table 2.3, 2.4(107) list an abridged comparison of these models. For detailed comparisons of more models, refer to (107).

3. DOODBMS Design Methodology: With the rapid progress of the current technology, a evolution or revolution in computer programming, hardware structure, algorithms, CASE tools, or design methodology may cause the research community restructure the design methodologies in that associated domain. There are many object-oriented software design methodologies; however, they take little or no account of databases(11). There are many DDBMSs (mostly assumed relational data model) and OODB design methodologies, however, there is no integrated design methodology for DOODBMS. This thesis attempts to provide a guideline for DOODBMS application developers by putting the above design methodologies together and deriving a methodology suitable in this domain. It is not reasonable to expect that application developers will discard the old methodologies. We must, therefore, devise new design methodologies for DOODBMS application developers by either generalizing the existing ones or by providing a migration path from the old to the new methodologies with considerations of current technologies.

4. DOODBMS user-interface development systems (UIDS): User-interface tools come in two general forms: user-interface toolkits and UIDSs. User interface toolkits were indicated having many shortcomings(110): difficult to use, difficult to understand and edit, too little functionality, hard to build, deep learning curve, etc. The problems of toolkits have led to the creation of UIDS. UIDS is a integrated set of tools that help application developers create and management many aspects of interfaces. Despite the success of UIDS, the area of user interface for database systems remains largely barren. Most of the work have been focused on the design of a specific user interface for a specific system, rather than the development of a generic user interface technology(11). In RDBMS, many platforms offer variations of SQL languages as their user interface. The primary shortage of SQL is that it requires that users

	1	2	3	4	5	6	7	8	9	10	11	12
1. OOA Process												
Problem Domain Analysis												
(a) Identification of: Semantic classes	X	X	X	X	X	X	X	X	X			X
Attributes		X	X	X			X	X				
Behavior	X	X	X	X		X	X	X	X			X
Relationships:												
Generalization		X	X	X		X	X	X	X			X
Aggregation		X	X				X		X			
Other		X					X	X	X			
(b) Placement of:												
Classes			X	X					X			
Attributes			X									
Behavior	X	X			X	X	X		X			
(c) Specification of: Dynamic behavior	X	X					X					
2. OOD Process												
Solution Domain Design												
(a) Identification of:												
Interface classes			X									
Base/Utility classes	X				X		X					
(b) Optimization of classes	X	X	X			X	X		X			

Legend:
1. Alabiso 2. Booch 3.Coad and Yourdon 4. Corman and Chooblneh
5. Kappel 6. Meyer 7. Rumbaugh et al. 8. Shlaer and Mellor
9. Wirfs-Brock et al. 10. Pages-Jones et al. 11. Wasserman et al.
12. Henderson-sellers and Constantine

Table 2.3. Comparisons of Some Major OOA and OOD Processes and Representations

	1	2	3	4	5	6	7	8	9	10	11	12
3. Representations												
(a) Static view:												
Objects	X	X	X	X	X	X	X	X	X	X	X	
Attributes	X	X	X	X	X	X	X	X		X	X	
Behavior	X	X	X	X	X		X	X	X	X	X	
Relationships:												
Generalization	X	X	X	X	X	X	X	X	X	X	X	
Aggregation		X	X		X		X					
Other		X			X		X	X			X	
(b) Dynamic View:												
Communication	X	X	X	X	X				X	X	X	
Control/Timing	X	X	X	X	X		X				X	
(c) Constraints:												
On Structure		X	X	X			X	X			X	
On Dynamic Behavior		X			X		X	X				
4. Complexity Management												
(a) For Structure Complexity		X	X			X	X	X	X			
(b) For behavioral complexity									X			
(c) Representation of:												
Static structure		X	X						X			
No. of issues Addressed	12	21	19	12	12	9	20	13	15	5	8	3

Legend:
1. Alabiso 2. Booch 3.Coad and Yourdon 4. Corman and Chooblneh
5. Kappel 6. Meyer 7. Rumbaugh et al. 8. Shlaer and Mellor
9. Wirfs-Brock et al. 10. Pages-Jones et al. 11. Wasserman et al.
12. Henderson-sellers and Constantine

Table 2.4. Comparisons of Some Major OOA and OOD Processes and Representations(cont.)

have some sorts of SQL knowledge in order to access data even though SQL is non-navigational or non-procedural[1]. In OODBMS, some existing platforms (such as ITASCA) offers interface to SQL-like object query and the absence of object query language standard force some modern application developers desires a new user interface for OODBMS. This interface is suggested to deal with the following(11):

(a) Complex objects and their representations on the screen.

(b) Multimedia objects.

(c) Actives objects.

(d) Display and manipulate very large objects on the screen.

(e) Add in distributed system user interface features considerations.

(f) Interactive user interface with quick response is desired in distributed systems.

These functionalities are suggested for either building or searching a UIDS for application design. A continent and easy-to-use user interface without much learning curve (such as graphical user interface or voice-activated commands) are desired to improve the convenience and efficiency, which are the primary goal of a DBMS(89).

2.6.2 DDBMS and OODBMS Unsolved Problems. Even after about a decade of efforts in DBMS, there are many unsolved problems(122), (84) in DOODBMS. Among them, network scaling problems, distribution design problems, distributed query processing, operating systems support, new hardware, etc..

1. Network scaling problems: the database community does not have a full understanding of the performance implications of all the distributed DBMS design alternatives. Specifically, questions have been raised about the scalability of some protocols and algorithms as systems become geographically distributed or as the number of system components increases. One concern is the suitability of the distributed transaction-processing mechanisms (the two-phase locking and, particularly, the two-phase com-

[1] The data manipulation languages require a user to specify wh.. data is needed and without specifying how to get it(89).

mit protocols) in distributed database systems based on wide area networks. Significant overhead is associated with these protocols, and implementing them over a slow wide area network may be difficult.

2. Distribution Design problems: Distributed database design methodology varies according to system architecture. For tightly integrated distributed databases, design proceeds top-down from requirements analysis and logical design of the global database to physical design of local databases. For distributed multidatabase systems, the design process is bottom-up and involves the integration of existing databases. The step of interest to us in the top-down process is distribution design, which involves designing local conceptual schemas by distributing global entities over the distributed system's sites. The global entities are specified within the global conceptual schema.

3. Distributed Query processing: Distributed query processors translate a high-level query on a distributed database, which is seen as a single database by users, into an efficient low-level execution plan expressed on local databases.

4. Parallel database systems: The design problems of parallel database systems, such as operating system support, data placement, parallel algorithms, and parallelizing compilation, are common to both kind of architectures. If parallel data servers become prevalent, we can foresee an environment in which several of the them are placed on a backbone network, giving rise to distributed systems consisting of processor clusters.

5. Operating Systems Support: The DBMS's requirements and the current operating systems' functions are mismatched. This disparity is even more true in DDBMS, which require functions that existing distributed operating systems do not provide, for example, distributed transaction support with concurrency control and recovery, efficient management of distributed persistent data, and more complicated access methods. Furthermore, DDBMS necessitate modifications in how the distributed operating systems perform their traditional functions (for example, task scheduling, naming, and buffer management). The basic issues in DBMS distributed operating system integration: system architecture, transparent naming of resources, persis-

tent data management, remote communications, and transaction support remain unsolved.

6. Multidatabase Systems: Multidatabase system organization is an alternative to logically integrated DDBMS. The main difference between the two approaches is the level of autonomy afforded the component data managers at each site. While integrated DBMSs' components are designed to work together, multi-DBMS consist of components that may have no notion of cooperation. Specifically, these components are independent DBMS, which means, for example, that although they may have facilities to execute transactions, they are incapable of executing distributed transactions that span multiple components.

7. New Hardware: Parallel Machines and Workstations. Parallel database systems are designed to exploit recent multiprocessor computer architecture to build high-performance and fault tolerant database servers. For obvious example, by fragmenting the data base across multiple nodes, we can obtain more inter-query and intra query parallelism. For obvious reasons such as set-oriented processing and application portability most of the work in this area he focused on supporting SQL. The design problems of parallel database systems, such as operating system support, data placement, parallel algorithms, and parallelizing compilation, are common to both kinds of architectures. If parallel data servers become prevalent, we can foresee an environment in which several are placed on a backbone network, giving rise to distributed systems consisting of processor clusters. The early commercial OODBMSS (for example, Servio Corp.'s Gemstone) use a client/server architecture in which multiple workstations can access the database centralized on a server, distributing an object-oriented database within a network of workstations (and servers) is becoming very attractive.

8. Lack of Standard Terminology: There is much terminology confusion in this area, such as the abbreviated notations in Figure 2.1, different experts may have different interpretations on the same term.

9. Authorization of OODBMS: the generalization and aggregation relationships and inheritance on the class hierarchy make it difficult to implement authorization of

OODBMS(84). ORION makes the first attempt to define an authorization model for object-oriented ABA semantic databases(84). ITASCA, a commercial version of ORION, currently supports authorization in a limited fashion (assume all the users are superusers). This may also explain why OODBMS is good for engineerings application—The Engineer's integrity is doubtless!

Some authorization questions(84) are listed as follows:

(a) the right to create a subclass of an existing class must be defined as a new type of authorization. This is different from an authorization to create a class, since creating a subclass means inheriting attributes and methods defined in one or more existing classes.

(b) authorization for nested objects need to be added.

(c) authorization for versionable objects is desired.

(d) It is not practical to treat an authorization on a class as a property of the class, so that subclasses of the class will inherit it. Because this implies that a user with an authorization on a class will have the same authorization on all subclasses. It is more reasonable to require the other users to grant the creator of the superclass appropriate authorizations on the subclasses.

2.7 An Approach to a DOODBMS Application Design

The analysis and design of data distribution requires user-supplied information to perform the distribution of a database schema on different sites. The objective of this design methodology is to evaluate the utility of partitioning data objects into fragments and then to determine the allocation of fragments to each site. The following sections are some major strategies that will be applied for designing application databases(30):

2.7.1 Top-Down Design Approach. In this approach, developers start by:

1. Performing a requirement analysis that defines the environment of the system to produce a design specification. The requirements study also specifies where the final system is expected to stand with respect to the objectives of a DDBMS,

2. Performing a view design and conceptual design (the design of the global schema). The view design deals with defining the interfaces for end users. The conceptual design, on the other hand, is the process to determine entity types and relationships among these entities. In conceptual design and view design activities, the users need to specify the data entities decide the applications that will run on the database and statistical information about these applications. Up to this point, the process is identical with that in the centralized database design,

3. Collecting access pattern information and global conceptual schema. This information is input to the distribution design step. The objective of this step, which is the focus of this thesis, is to design the local conceptual schema by distributing the entities over the sites of the distributed system,

4. Partitioning database to proper fragments. Rather than distributing relations, it is quite common to divide them into fragments. The distribution design consists of two steps: fragmentation and allocation,

5. Designing the physical schema. This objective of this step is to map the local conceptual scnema to the physical storage devices available at the corresponding sites,

6. Monitoring the whole design process of the behavior of the database implementation and the suitability of users' views. (A process of top-down design approach is shown in Figure 2.2)(121).

This approach is the most attractive for systems that are developed from the beginning without connecting with any existing database(30). Yet, this approach is very difficult to apply in aggregating heterogeneous databases that already exist. The main problem with the top-down approach to distributed database is that there are usually many sites, each with its own DBMS and data already in place. What is needed is a layer on top to provide transparent access to independent database that exist at multiple sites. Instead, we apply the bottcm-up approach as an alternative.

2.7.2 Bottom-Up Design Approach. When we want to integrate different existing databases, the bottom-up approach can be applied. This approach is based on the integration of existing schema into a single, global schema(30). By integration we mean

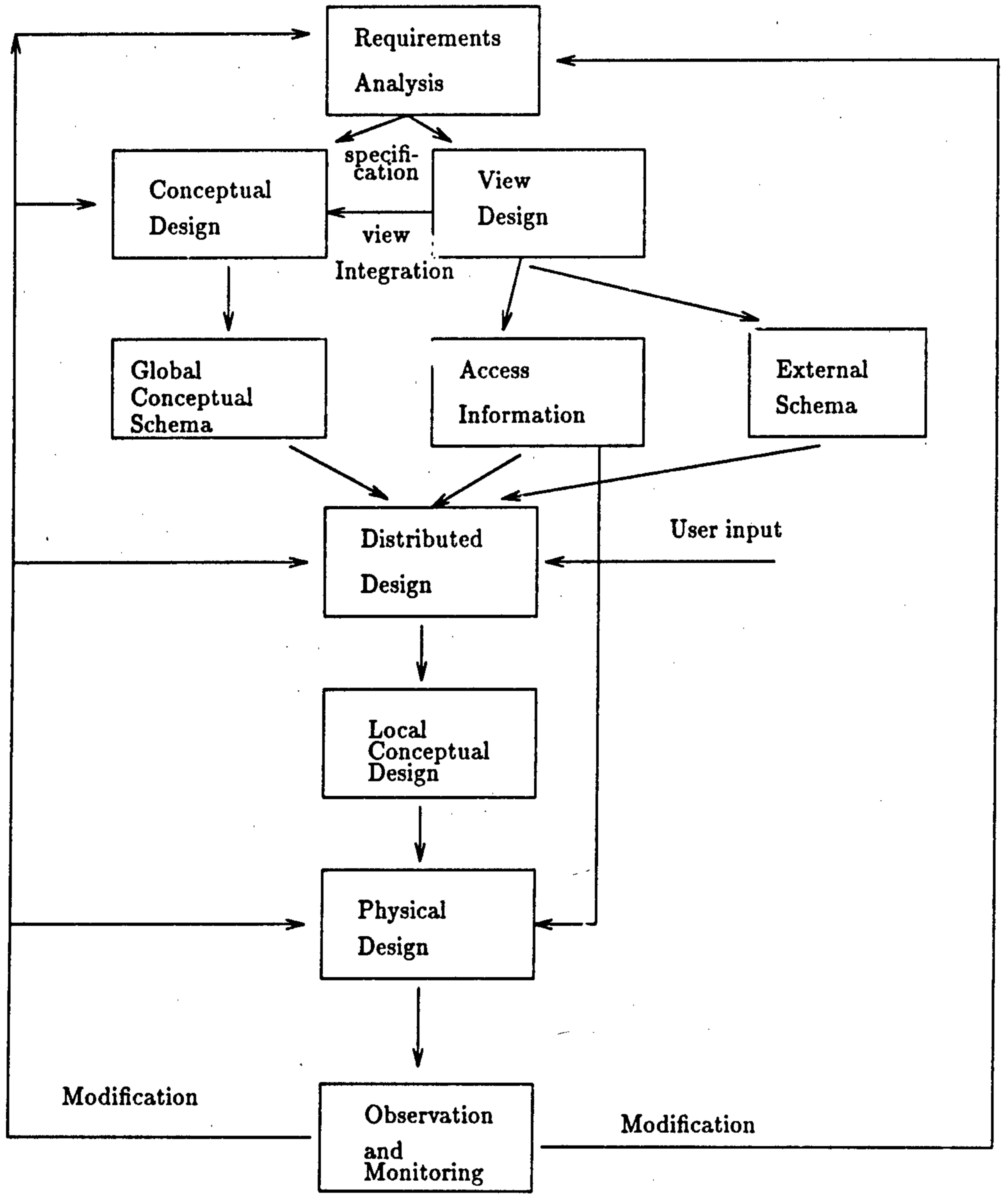


Figure 2.2. A Framework for Distributed Database Design

the merging of common data definitions and the conflicts among different representations given to the same data. For heterogeneous multi-database integration, this approach is appreciated. This bottom-up approach is called a federated DBMS. One result of a federated DBMS is that greater local autonomy is preserved. In a distributed environment, each participating database needs to be able to define its data independently of the other sites. There may be small differences, such as using different names for the same entity, or different lengths for a character string.

Another requirement for local autonomy is that the local DBMS must be able to operate independently of the distributed DBMS. For example, the DBMS should be able to determine the hours of operation, when it is going to perform a backup, and how to recover from a system crash. However, many integration methods must be applied, i.e., schema translation and schema integration(121).

In the real world, both design approach should be applied to design DDBMS, since no single design approach can fit into the real world perfectly. Especially in the design process of integrating the multi-database systems, the individual databases that applied a top-down approach already exist and the integration of these databases must result to bottom-up approach. Bottom-up methods apply schema translation and integration that combine those individual databases into one multi-database. Besides, object-oriented design is basically a bottom-up approach. Simple objects can be implemented and tested before being used.

2.8 Prototyping

The sequence of events for the prototyping paradigm is illustrated in Figure 2.3(132). This paradigm begins with requirements gathering: developer and customer define the overall objectives for the software. These overall objectives can be in the form of memc describing a problem, a report defining a set of product goal, or a system specification(132). Then, a quick design occurs. Prototypes typically use the following tools to achieve a quick design: 1) integrated data dictionary, 2) screen generators, 3) non-procedural report writer, 4) 4GL query, and 5) DBMS(182). After applying the tool(s), the prototype can be built. The reason for prototyping is to help discover missing requirements and test candidate

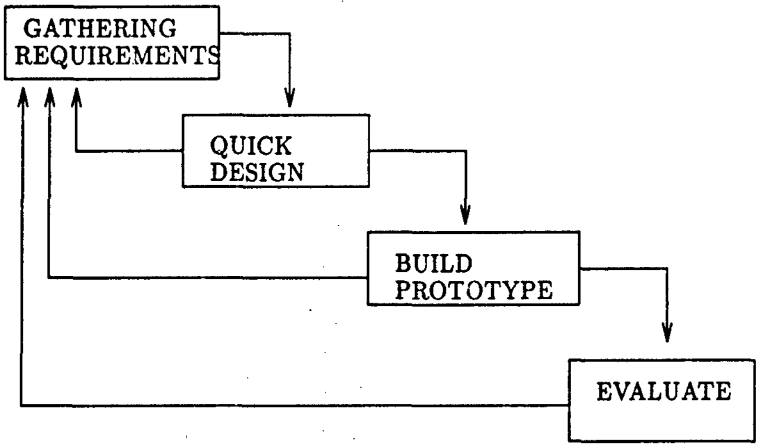


Figure 2.3. Prototyping Paradigm for a Database Application Design

design. The prototype is evaluated by the user and is used to refine requirements for the software to be developed. This process is iterative until the software product satisfy the need of the users.

2.8.1 OODBMS Features that aid rapid prototyping. Several OODBMS features help us build the application prototype. These features are as follows:

1. Modeling power: One majɔr advantages of the object-oriented paradigm is increased modeling power. The objects and their interrelationships in OODBMSs can be aligned very closely to the real world objects and their interrelationships. This function can be achieved by using the features of object-oriented modeling: object specialization, object generalization, classification, aggregation, data abstraction and data hiding.

2. Code reusability: Object-oriented programming languages allow the programmers to write less code and implement the same functionality by reusing the code modules, for instance, predefined routines that already exist on the system. Moreover, inheritance enables the attributes and functions from superclasses to their subclasses. Thus, a programmer does not need to reimplement the functions that a subclass shares with its superclass. This enhances the code reusability.

2.8.2 Database Design Considerations.

In general, the goal of a DBMS design is to generate a set of schemes that allow storing information without unnecessary redundancy, yet allow easy retrieval. A badly designed database scheme may have the following undesirable properties:

1. Repetition of information,
2. Inability to represent certain information,
3. Loss of information.

To achieve a good design, these undesirable properties must be avoided and also need to minimize the data access speed and storage costs.

2.9 Object-oriented Paradigm

Developing an effective DOODBMS application is a difficult task, because there are many competing requirements. We should not only satisfy the functional requirements of the application, but also address time and space factors. A time-inefficient database that retrieves data long after it is needed is useless. Similarly, a database that requires a building full of computers and a swarm of people to support is not cost-effective. To achieve time-efficiency and cost-effectiveness of our application database, we should perform domain analysis carefully. Domain analysis seeks to identify the classes and objects that are common to the application.

Generally speaking, there are three phases to the software life cycle in object-oriented paradigm: 1) analysis, 2) design, and 3) implementation(61). Most of the software development methodologies also have these three phases. However, in contrast to more traditional methods, the object-oriented paradigm is iterative and incremental(23).

Henderson-Sellers et al.(61) propose a fountain model to illustrate that the object-oriented version of the overall software development life cycle does have the characteristics of high degree of overlap and iteration. This model provides a diagrammatic version of the stages present in object-oriented software life cycle and a clear representation of the features of object-oriented technology life cycle.

2.9.1 Major Elements of the Object-oriented Paradigm. Booch(23) suggested that there are four major elements in the conceptual framework of object-oriented modeling. These principles are: abstraction, encapsulation, modularity, hierarchy. This modeling technique is based on the OOP languages. Object-oriented modeling is a technique that emphasizes the principles of object-oriented desig.ı. These principles are: abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence(23). Abstraction denotes the essential characteristics of an object that distinguish it from all other kind of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential c. aracteristics(23). Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules(23). Hierarchy is a ranking or ordering of abstractions(23). Hierarchy allows single or multiple inheritance of objects. Inheritance is the technique to define a hierarchy among classes in which a subclass inherits from one or more superclasses. A class is a set of objects that share a common structure and a common behavior. Subclass is a class that inherits from one or more classes. A superclass is a class from which another class inherits. There are basically two types of relationships between classes: inheritance and composition.

Inheritance is used to describe Generalization and Specialization (Gen-Spec structure)(34). Inheritance can be referred to as a "is a" or "is a kind of" relationship between class. For example, an "employee" (subclasses) can be thought of as a "person" (superclass). The subclass is more specific than the superclass that it is derived from. An employee object will have the same attributes and services that a person object has in addition to attributes and services that are common only to employee.

Composition is used to describe "Whole-Part" relationship(34). A simple example would be that a vehicle consists of an engine, a transmission system, and a brake system, where vehicle, transmission, brake, and engine are distinct classes and objects(34). Typing is the enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways. Concurrency is the property that distinguishes an active object from one that is not active. Persistence is the property of an object through which its existence transcends time and (or) space. Booch (23) states that software design is an iterative process that requires broad and deep skills.

This modeling technique is fundamentally different from the traditional modeling technique. For instance, In a value-based data model such as the relational model, data is identified by values. This way of identifying data leads to several problems. One major problem is that the modeling of relationships among data leads to data redundancy(121) (use of foreign keys). An object identifier is the primary key for an object (unique object identifier is enforced, therefore, no redundancy). The trade-off is referential integrity checks have to be programmed into the methods that update or delete objects(8). Another important problem is the lacking of precise semantics for updates(121).

The foregoing problems disappear in the object-oriented model because this model adopts a natural integration with OOP languages that is made possible by object identity and the capability of modeling arbitrarily complex data structures. However, Object-oriented design is a young practice, a discipline for effectively applying the elements of the object-oriented model has not yet been defined.

2.9.2 Object-oriented Analysis (OOA). Object-oriented analysis (OOA) is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain(23). Booch (23) and Rumbaugh et al(144) recommended trying to identify classes and objects from the nouns in the problem statement. The purpose of object-oriented analysis is to model the real world system so that it can be understood(34). This analysis focuses on the "what" of the system instead of the "how". In object-oriented analysis, we build requirements models that expose the existence and behavior of entities, entity types, relationships, and relationship types, along with events in the user's world that affect these things.

During the literature research, the author found that the boundary between OOA and OOD is inconsistently defined in the literature. Some processes used by one author during OOA maybe included in another author's OOD technique. For example, Coad and Yourdon(34) mention state-transition diagrams and other dynamic representations as part of OOA, whereas Booch(23) discusses views of the system in OOD. Rumbaugh et al. (144)

Steps proposed	Sources
Identify Classes and Objects	Booch – Step 1 Coad, Yourdon: Object Layer Step 1 Martin, Odell: Object Struct., Behavior Analy. Step 1, 2 Rumbaugh et al: Object Model Step 1
Identify Attributes	Booch : Step 1, 2 Coad and Yourdon: Attribute Layer Step 4 Martin, Odell: Object Structure analysis Step 1, 2 Rumbaugh et al: Object Model Step 4
Identify the Structure (Relationship)	Booch : Step 4 Coad, Yourdon: Structure Layer Step 2 Martin, Odell: Object Structure analysis Step 2, 3 Rumbaugh et al: Object Model Step 3,5,and 6
Defining the Services	Booch : Step 3 Coad, Yourdon: Service Layer Step 5 Martin, Odell: Object Behavior Analysis Step 4,5, 6, 7 Rumbaugh: Object Model Step 3,5,6

Table 2.5. Comparisons of Some Proposed OOA Activities

use structural, dynamic and functional representations during both analysis and design. The activities in OOA proposed by many experts are similar in the following four steps: 1) Identify the classes and objects of the problem domain, 2) Identify the attributes, 3) Identify the Structures (relationships), and 4) Defining the services. Table 2.5 shows a list of the steps related to these four steps.

2.9.2.1 Data Dictionaries (DD). Words, too often, have many interpretations, so desired for all modeling entities. Rumbaugh et al(144) proposes the following ways of defining a data dictionary: 1) writing a precisely describing each object class, 2) the class within the current problem, 3) describing attributes, and operations. The Data Dictionaries (DD) is an important part of the requirement specification. Without DDs, an analysis of the information domain would be incomplete. Table 2.6(132) shows these common notations that have been used to define traditional DD in this thesis.

2.9.3 Object-oriented Design (OOD). During analysis, the focus is on what needs to be done instead of "how". During design, decisions are made about how the problem will be solved, first at a high level, then at increasingly detailed levels(144). After

NOTATION	MEANING
+	and
=	is composed of
( )	enclosed component is optional data
{ }	enclosed component is iterative
[ ]	select one of the options enclosed in the brackets
* *	comment
:	separates alternative choices in the [ ] construct

Table 2.6. Notations of Data Dictionary for Attributes

OOA has been done, Coad and Yourdon suggested OOD can be achieved based upon the construction of the four components: 1) the Problem Domain Component, 2) the Human Interaction Component, 3) the Task Management Component, and 4) the Data Management Component(34). Coad and Yourdon(34) offer a number of criteria to use when adding the OOA results to the construction of these four components.

Also, methodology of Object-oriented design from Rumbaugh et al(144) have been applied. As they pointed out in the "Steps of Object Design"; Determine object representation, Adjust class structure to increase inheritance, etc. Object-oriented design, as Booch (23) suggested, is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design. The object-oriented decomposition is based upon objects, not algorithms. This decomposition has some highly significant advantages over algorithmic decomposition as in top-down structured design. Object-oriented decomposition yields a smaller system through the reuse of common mechanisms, thus providing an important economy of expression(23). Object-oriented systems are more resilient to change than structured-design systems, thus better evolve over time. Also, the object-oriented design reduces the risk of building complex software systems, because they are designed to evolve incrementally from smaller systems to large systems.

OOA and OOD are solutions for narrowing the chasm between analysis and design. In the past years, software developers have been suffered as they tried to move the analysis results directly to design. This prevents designers from systematically adding design-dependent detail to the analysis results. However, OOA helps identify and define Classes and Objects that directly reflect the problem domain and system's responsibilities within it. OOL helps identify and define additional Classes and Objects that directly reflect an implementation of the requirements(35). By applying a uniforın representation for organizing data and behaviors removes the chasm between analysis and design. These activities proposed by some well-known experts are shown in Table 2.7.

2.9.4 Object-oriented Programming (OOP). As Stroustrup(160) pointed out in "What is Object-Oriented Programming?":

> A language is said to support a style of programming if it provides facilities that make it convenient (reasonably easy, safe, and efficient) to use that style.

Wolf(176) suggested that a language is object-criented if it supports the following three mechanisms without exceptional effort or skill to write such programs:

1. data abstraction,
2. inheritance,
3. and runtime method determination.

Because of the impact of Object-oriented Programming (OOP) languages, object-oriented design has become one of the most popular design methodologies in software engineering and database design. This programming paradigm subsumes the concept of abstract data types (ADT) that define a public and private portion of data structure called object(28). The idea of ADT is to group together a set of data values and the set of operations on those data values. ADTs offer excellent means of separating the functicnality from the implementation of the data type. Direct access to the data is impossible. The only way to update, retrieve the data is by means of operations. These operations are functions or procedures that interact with the ADT in a well defined manner. The functionality of the operations can be specified by pre- and post-conditions(152). There are two advantages of using ADT: 1) the users of ADTs have a conceptually simple interface to use ADTs without knowing the implementation details of ADTs. 2) Changes of ADTs do not affect the pre- and post-conditions of ADTs, and therefore have no consequences for the users(152). There are three important characteristics in OOP: 1) OOP uses objects, not algorithms, as its

Sources	Steps proposed
Booch	1. Identify the classes and objects at a given level of abstraction. 2. Identify the semantics of these classes and objects. 3. Identify the relationships among these classes and objects. 4. Implement the classes and objects.
Coad and Yourdon	1. Problem Domain Component (PDC) design. 2. Human Interaction Component (HIC) design. 3. Task Management Component (TMC) design. 4.Data Management Component (DMC) design.
Lorensen	1. Identify the data abstraction for each subsystem. 2. Identify the attributes for each abstraction. 3. Identify the operations for each abstraction. 4. Identify the communication between objects. 5. Test the design with scenarios. 6. Apply inheritance where appropriate.
Martin and Odell	In Both Object Structure and Behavior Design, do the following: 1. Identify classes to be implemented. 2. Identify structure of each class. 3. Identify operations each class offer. 4. Identify how the class inheritance be implemented. 5. Identify the variants.
Rumbaugh et al	1. Combine the three models to obtain operations on classes. 2. Design algorithms to implement operations. 3. Optimize access paths to data. 4. Implement control for external interactions. 5. Adjust class structure to increase inheritance. 6. Design associations. 7. Determine object representation 8. Package classes and associations into modules.

Table 2.7. Some Proposed OOD Activities

fundamental logical building blocks; 2) Each object in OOP is an instance of some class; 3) classes are related to one another via inheritance relationship. So, any programming language is object-oriented programming language if and only if it satisfies the above three characteristics.

2.9.4.1 Object-oriented Databases Programming Languages (OOPLs). Traditional approach defined a database language to be embedded in host programming. The problem with this approach is that the application programmers have to learn and use two different languages. Furthermore, the application programmers have to negotiate the differences in the data models and data structures allowed in the two languages.

Another favorable approach is to extend object-oriented programming languages with database-related constructs. This approach makes it possible for the application programmers to learn only new constructs of the same languages, rather than an entirely new language. this approach is more desirable, unless, for example, application written in more than one language must share a common database(84).

OOPLs such as Smalltalk, C++, Objective-C, Eiffel, ODE[2], Object Pascal, Actor, Lisp (CLOD), Ada (9x), Traits, Trellis/Owl, Flavors, Object LISP, Oaklisp, LOOPS, Common LOOPS, Classcal (an extension of Pascal) etc.. are some well-known languages. Table 2.8(22), (144), (97), (2) lists a brief comparisons.

In DoD software development, Ada has been mandated as the DoD language of choice based on a serial of researches performed by various organization (such as Institute for Defense Analyses (IDA), Software Engineering Institute, CTA, and Naval Postgraduate School). These research results reached the same conclusion: there are no compelling reasons to waive the Ada requirement to use C++ (2). Table 2.9(3), (125) and Table 2.10(2)[3] list comparisons of the capabilities and language's features of Ada and C++. Ada does have more scores and + signs than C++, however, the lack of multiple inheritance and polymorphism make Ada a language that must keep improving. For the features that only specific to Ada, C++ does not have those features because of those functions are

[2] A database extension to C++. It extends C++ to provide persistence, versioning, constraints and triggers, query processing constructs(64)

[3] Both Tables are derived from the information provided in (2).

Capability/Feature	C++	Lisp (CLOS)	Smalltalk	Eiffel	ODE	Ada (9x)
Database	No	No	No	Yes	Yes	No
Window Toolkit(GUI)	Yes	No	Yes	Yes	Yes	Yes
Strong type Checking	Yes	No	No	Yes	Yes	Yes
Standard Class Library	No	No	Yes	Yes	Yes	Yes
Multiple Inheritance	Yes	Yes	No	Yes	Yes	Not Yet
Garbage Collection	No	Yes	Yes	Yes	Yes	Yes
CASE Tool Support	Excellent	No	Good	Good	Good	Good
Dynamic Binding	Yes	Yes	Yes	Yes	Yes	Yes
Static binding	Yes	No	No	Yes	No	Yes
Metadata at run-time	No	Yes	Yes	No	Yes	Yes
OS support	Yes	Yes	Yes	Yes	Yes	Yes
Object-Oriented	Yes	Yes	Yes	Yes	Yes	Soon
Low Level Expressiveness	Excellent	Good	Fair	Good	Good	Fair
High Level Expressiveness	Fair	Very Good	Good	Good		Good
Popularity	excellent	Fair	Good	Fair	Fair	Fair
Maintainability	Fair	Fair	Good	Good	Good	Very Good
Training	Good	Fair	Fair	Fair	Fair	Good

Table 2.8. Comparisons of Some Major OOPLs

already supported by the Unix Operating System (OS) (functions such as Concurrency control, Compilation Management, External interrupts, Interprocess communication etc.). However, this OS support can be system-dependent, different OS, hardware may have different supports. For only a comparison of language features is not explanatory in the software engineering capabilities.

It is more helpful and meaningful to classify in multiple dimensions for comparing how well these two languages can attain the goals of software development support (such as: reusability, expedient development, modifiability etc.) The results of comparing these two languages in the respect of of supporting software engineering are listed in Table 2.10. For a detailed explanations of features where Ada has an advantage or features where C++ has an advantage, refer to (2).

2.9.5 CASE Tools Supporting Object-Oriented Software Development. During the literature research, some vertical tools[4] were found to support object-oriented analysis,

[4] Vertical tools support specific activities in a single life cycle phase, like analysis, design, or testing(155).

Language Features	Ada Only	Ada +	Both =	C++ +	C++ Only
Concurrency	X				
Error Handling	X				
External Interrupts	X				
Parameterized Types	X				
Safe Types	X				
Modifiability		X			
Readability		X			
Compilation Management		X			
Data Abstraction			X		
External I/O			X		
Extensible Typing			X		
Information Hiding			X		
Modularity			X		
Overloading			X		
Strong Types			X		
Multilingual Support				X	
Single Inheritance				X	
Conditional Compilation					X
Dynamic Binding					X
Multiple Inheritance					X
Polymorphism					X
Subprogram Variables					X

Legend:

Ada Only : a Language Features that only ADA has.
Ada + : a Language Features that ADA is considered more powerful.
Both = : Both languages are equal.
C++ + : a Language Features that C++ is considered more powerful.
C++ Only : a Language Features that only C++ has.

Table 2.9. Comparisons of the Key Language Features of Ada and C++

Capability	Ada	C++
Reliable S/W Engineering	4.5	3.2
Maintainable S/W Engineering	4.4	3.2
Reusable S/W Engineering	4.1	3.8
Realtime S/W Engineering	4.1	2.8
Portable S/W Engineering	3.6	2.9
Runtime S/W Engineering	3.0	3.6
Compile-time Performance	2.3	3.1
Multi-lingual Support	3.1	2.4
OOD/Abstraction Support	3.9	4.6
Program Support Environment	4.1	2.1
Readability	4.4	2.9
Write-ability	3.4	3.5
Large Scale S/W Engineering	4.9	3.3
Cost S/W Integration	2.8	3.6
Precedent Experience	3.6	1.5
Popularity	2.8	4.0
Existing Skill Base	3.0	1.8
Acceptance	2.5	3.3
Total Score for MIS	1631	1324
(Ada is 23 percent higher)		
Total Score for C3 Systems	1738	1401
(Ada score is 24 percent higher)		

(Note: 0 = no support; 5 = excellent support)

Table 2.10. Comparisons of the Software Engineering Capabilities of Ada and C++

design and programming (such as OMTool, OOATool). These tools provide automatic process of preparing and updating object-oriented models, adding implementation details and generating data structures and class descriptions in object-oriented languages. These tools also defines relationships, aggregation and generalization, moves diagram elements while maintaining element relationships. Export diagrams or drawings to Postscript, Interleaf or Framemaker. Some horizontal CASE tools[5] (such as Teamwork/Object-Oriented Design). These tools provide a bridge between design and implementation in object-oriented environment. Most of them consist of graphical editor and C++ code framework generator, interface to certain C++ preprocessor (such as Centerline's ObjectCenter C++ compiler, AT&T C++).

2.10 An Approach for Data Distribution Design

When the designers design a centralized database, they design the conceptual schema that describes the integrated database. Next, they design the physical schema by mapping the conceptual schema to storage areas and determining appropriate access methods. For a distributed database design, the above two schema become the design of the global schema, and the distribution of the data to each site is derived in the following two steps:

1. The design of fragmentation that determines how global objects and classes are divided into fragments of objects (i.e., new objects, superclasses, subclasses.)
2. The allocation of fragments that determines how fragments are mapped to physical images; in this way, also the replication of fragments is determined.

2.11 Data Fragmentation and Replication

From a data distribution viewpoint, there is no reason to fragment data. Because, even in distributed file systems, the distribution is performed on the basis of entire files. However, with this centralized approach, the entire database is stored at a single node. This used to be the only approach, but now is only one of several possible approaches in

[5] Horizontal tools support activities across the entire life cycle, like project management and cost estimation(155).

distributed system design. Because there is only one copy of the database, it will certainly minimize the storage cost and the update costs. The advantage of this kind of system is simplicity.

However, there is a serious problem: single failure of the node, either the communication network, database or the hardware of the local computer, will cause the failure of the whole system. Moreover, the communication costs and retrieval costs are very high and the overall performance is very low. Data fragmentation and replication play important roles in a DDBMS; they solve the problems that we mentioned above thus improve the performance of DBMS. The design of fragmentation is the first problem that must be solved in the top-down design of data distribution(30). The relations in the DDBMS schema are usually decomposed into smaller fragments. But how do we decide the degree of fragmentation? What's the proper way to insure that the fragmentation is correct? Most the literatures in this area suggest the follows:

2.11.1 Global Schema and Fragmentation Schema. The global schema are objects we defined for the application database. Let's assume that there are three sites that receive the data provided these three locations are equipped with equivalent capability of computers and the distance between each two locations is equal. Our physical design, based on the following schema, will be well-considered.

In relational DDBMS, there are three ways of dividing a relation table into a smaller one: dividing it horizontally, vertically or both(30). The same principles can be applied to DOODBMS by extracting the attributes from the objects then 1) declare the attributes shared, 2) create a new version, or 3) create a new class. These three alternatives involve trade-offs in retrieve, update cost and user access information.

1. Horizontal Fragmentation: Horizontal fragmentation partitions objects along their attributes. Thus each fragment has a subset of the attributes. To achieve a proper design of horizontal fragmentation, it is important to note: 1) database information: how the database relations are connected to one another, especially with joins. 2) application information: how the users use the application, both qualitative and quantitative information is required about applications. The quantitative informa-

tion is incorporated primarily into the allocation models, whereas the qualitative information guides the fragmentation activity. The fundamental qualitative information consists of predicates used in users queries.

There are two versions of horizontal partitioning: *primary* and *derived*(121): 1) Primary Horizontal Fragmentation: The horizontal fragmentation of a relation is based on a property of its own attributes(30). This fragmentation can be defined by expressing each fragment as a selection operation on the global relation, 2) Derived Horizontal Fragmentation: In some cases, the horizontal fragmentation of a relation cannot be based on a property of its own attribute but is derived from the horizontal fragmentation of another relation(30). To perform a derived horizontal fragmentation, three inputs are needed: 1) the set of partitions of the owner relation, 2) the member relation, and 3) the set of semi-join predicates between the owner and the member(121).

2. Vertical Fragmentation: Since horizontal partitioning subdivides object instances (tuples) into groups, the vertical partitioning (VP) is a methodology that subdivides attributes into groups and assigns each group to a physical object(113). Vertical partitioning is basically an empirical design of fragments based on the specified logical access frequencies of the transaction(113). The major information required for vertical fragmentation is related to applications. Since vertical fragmentation places in one fragment those attributes usually accessed together, this togetherness is called affinity. Vertical fragmentation is a heuristic and intuitive design optimized by the cost factor and performance. An overall approach to solving the vertical partitioning problem is shown in Figure 2.4(113).

Vertical partitioning is used to improve the performance of transactions because the smaller the fragment, the fewer pages in secondary memory are accessed to process a transaction. In order to improve performance, vertical fragmentation must be well-constructed. Let's consider some methodologies: 1) Vertical Partitioning with Overlap of Superkey: In general, this inclusion of a key of the global relation to the partitioned fragments is the most straightforward way to guarantee that the reconstruction through a join operation is always possible. The trade-off is that we

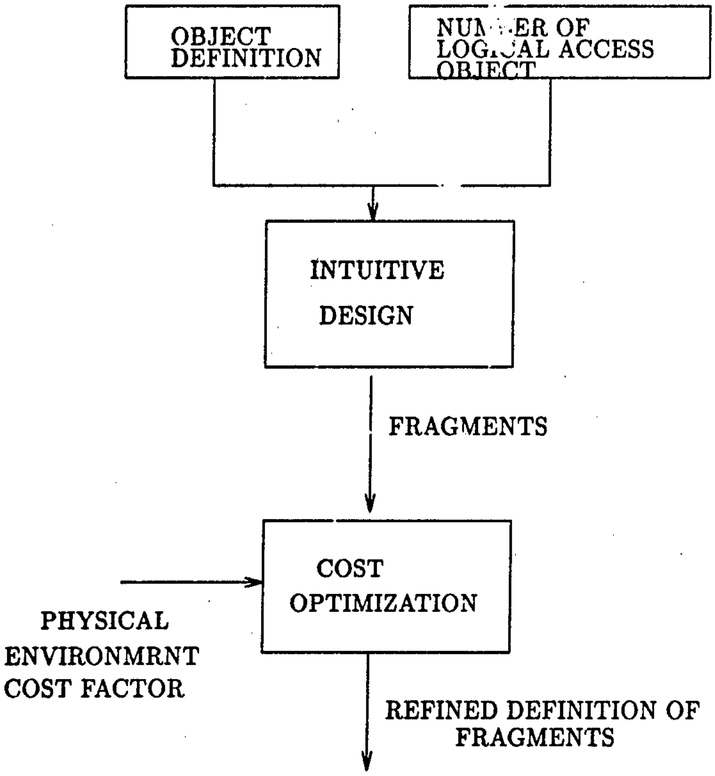

OBJECT DEFINITION
NUMBER OF LOGICAL ACCESS OBJECT
INTUITIVE DESIGN
FRAGMENTS
COST OPTIMIZATION
PHYSICAL ENVIRONMRNT COST FACTOR
REFINED DEFINITION OF FRAGMENTS

Figure 2.4. An Overall Approach to Solving the Vertical Partitioning Problem

include a redundant attribute in the partitioning schema, 2) Vertical Partitioning with a Tuple-id: More generally, vertical fragmentation is accomplished by adding a special attribute called a tuple-id to the relation(89). This is a changed version of vertical partitioning with overlap of superkey, and 3) Binary Vertical Partitioning (BVP): A BVP format is claimed to be non-overlapping that means the interception of the partitioned fragments is an empty set(113). BVP is ideally the best vertical fragmentation method that guarantee best storage, but not necessarily the best performance because it will be more complicated to join the data together.

3. Mixed Fragmentation: In the process of partitioning fragments, sometimes it is necessary to mix both horizontal and vertical fragmentation to produce a mixed relation to meet the need of the local physical design. Vertical fragmentation can significantly decrease the attributes when join operations are needed to produce network trans-

parency. However, providing this network transparency may be very costly since it involves many joins of data from each site. The designers may use the vertical fragmentation technique to avoid those attributes that are not necessary for SELECT operations. This can be achieved easily by PROJECT operation using SQL. However, to decide which attributes to P?OJECT, we need to apply to the vertical partitioning technique that we have mentioned above.

4. Replication: Replication is the allocation of a single fragment to multiple sites. Typically, replication is introduced to increase A&R of the system. When a fragment is not available due to site failure, it will be possible to access from another copy. The trade-off is in the cost of maintaining mutual consistency between the duplicated copies in multiple sites. For example, updates of a replicated fragment have to be made on all the sites having that fragment. Therefore, the update of a local copy requires an additional overhead for transmitting the update and ensuring that the update takes place on all of them. After partitioning the data, we need to define the distributed access plan for network transparency. How do the designers do the queries from the different sites. One simple way as mentioned before in this thesis is to create view as a unit of fragments. However, to provide this network transparency may be very costly since it involves many joins of data from each site. A comparison of replication alternatives is shown in Table 2.11(141), (121). This Table compares four replication alternatives with respect to various DDBMS functions, such as query processing. To achieve efficient data management, the importance of fragment allocation must be emphasized.

2.12 Allocation of Fragments

After the database files have been partitioned horizontally and (or) vertically, one critical distributed database design problem is how to assign file fragments to different sites for querying, updating, and execution with the conditions of satisfying the system objectives, such as R&A, organizational need and best performance. On the surface, object-oriented design seems ideally suited for distributed systems because object consists of data and control. Just as objects are used as the decomposition unit, the designers can also use

Functions	Centralized	F. R.	P. R.	P. P.
L. A.	N/A	Yes	Yes	Yes
R/A	low	high	high	high
Performance	low	high	high	high
C. I.	N/A	Yes	Yes	Yes
D. T.	N/A	Yes	Yes	Yes
C. T.	N/A	Yes	Yes	Yes
Q. P.	Easy	Easy	Difficult	Difficult (same as P. R.)
D. M.	Easy	Easy	Difficult	Difficult (same as P. R.)
C. C.	Easy	Moderate	Difficult	Easy
Reliability	Very Low	Very High	High	Low
Reality	Not Practical	Possible application	Realistic	Possible application
Storage Cost	Very Low	Very High	High	Low
Retrieval Cost	Very High	Low	Moderate	High
Update Cost	Very Low	Very High	Moderate	Low
Comm. Cost	Very High	Low	Moderate	High
Maintenance Cost	Low	Moderate	High	Moderate

Legend:
L. A. : Local Autonomy
R/A : Reliability and Availability
C. I. : Configuration Independence
D. T. : Data Transparency
C. T. : Control Transparency
P. R. = partial Replication
F. R. = Fully Replication
P. P. = Pure Partition (no replication)
Q. P. = Query Processing
C. C. = Concurrency Control
D. M. = Directory Management
N/A = Not Applicable
Note : pure partition means the database is divided into non-overlapping fragments.

Table 2.11. Comparisons of Replication Alternatives

them as the unit of distribution. Looking deeper, the issues are more complicated. This complicated problem is commonly known as the "file assignment problem" (FAP). Before modeling FAP solutions, we need to obtain the quantitative data about: 1) the database, which is the number of tuples of files that need to be accessed for a certain process and the size of the fragments, 2) the number of read accesses and update accesses of certain fragment in the application that run on the database, and 3) the communication network information which is queuing delays, cost of transmissions, network reliability, and overall system performance, 4) the site information which is storage capacity, performance of the local computer, cost of maintenance, etc. The file assignment problem is complex, and its solution is not trivial. For example, in a network of three nodes and forty files (objects) to share then the number of possible file assignments, where each node may have a copy of each file, is $2^{3^{40}}$ possible combinations. If there are no replicated copies, there are 3^{40} possible assignments. Thus, how to allocate the fragments properly becomes important to us.

2.12.1 FAP Objectives. The optimal distribution of file fragments among network storage nodes is one major problem in DOODBMS. Different design goals and varying system assumptions yield different FAP solutions. An acceptable FAP solution assigns files to different sites in some optimal fashion. This "optimal" fashion varies depending upon what the designers want to be designed in the system not upon how the designers want to design the system. There are two main considerations(43):

1. Minimum Cost: One major consideration is cost. The cost consists typically of the file storage costs, query costs, file update costs, and communication costs.
2. Maximum Performance: Another consideration is performance. Minimum execution and transfer time and maximum system throughput are two of the common performance objectives. These file assignment problems account for queuing delays and random routing capabilities. Also, if a file assignment is found that induces the optimal queuing network model parameters, then the file assignment is also considered optimal.

These two measures of optimality, cost and performance, can hardly coexist. Objective functions that minimize cost, such as storage costs and communication costs, are important variables that directly influence the initial configuration. Objective functions that optimize performance are applicable at production time. After the sy^stem is designed and the hardware is obtained, the cost becomes a secondary issue. The emphasis changes to performance. However, for a good FAP solution, both factors must be considered. Many approaches have been proposed by many experts in this field, such as Chu(32), and Ceri et al(30).

Chu(32) proposed various models that consider data sharing, maximum performance, minimum cost etc.. He experimented the problem of optimally allocating files in a computer network. He assumed that files may be stored in one or more node(s) that are commonly required by several nodes. Chu also assumed that the access rates are known. The FAP can be solved by a linear programming approach that yielded an optimal solution: a minimum overall operating costs (transmission and storage costs).

Ceri et al(30) also proposed a FAP solution for non-redundant allocation of the files. However, this model assumed that the computer network was completely connected to every node and each node can store multiple files. Before using this model, designer must have already collected the users requirements, derived the global schema, tabulates the expected load of transactions (query and update), and determined the distribution requirements (how the users would like the data to be distributed). With these as inputs, Ceri et al developed an optimal file allocation model. For this model, a linear programming technique was applied to minimize the overall operating costs. As the models proposed by Chu, this model has numerous variables that make it difficult to understand. Having studied the above FAP approach, it became clear that these models exhibited serious shortcomings:

1. These models used too many assumptions, which make these models not feasible to apply in the real world applications.
2. The nature of distributed environments is specific to applications, which required specific hardware architecture and software configurations.

3. numerous variables are required in each model, even for a network with a few nodes.

4. These models are research-oriented than application-oriented.

5. These models are difficult to understand and follow.

For these reasons, the above FAP solutions for local database design were not applicable for the design of the thesis database application. However, some principles were used to help achieve better local physical database design.

2.13 Summary

DOODBMS is one of the current trends in the database systems development, especially, when 1) organizations tend to be spread among a large area , 2) users in each site have the need to ensure the information is reliable and available at low cost, 3) users need to achieve a higher local autonomy, and 4) users wants to interconnect existing database systems with a higher performance, 5) complex data model is desired to efficiently manage data. However, along with these benefits come several disadvantages, including software development costs, greater potential for bugs, and increased processing overhead(89). The design of DDBMS begins when users propose the requirements of a system, the designer (analyst), together with the users, develop a system specification. Then the conceptual design and view design begin. Many commercial DBMSs are available and these systems contain a variety of query languages for data manipulation and access, 4GLs for applications development. Nevertheless, DOODBMS applications are still rare, thus designing a DOODBMS application is worthwhile.

III. Methodology

3.1 Introduction

In the past, designing and implementing a DBMS application was difficult and time-consuming. Recent advances in DOODBMS (such as DBMS platform, OOA, OOD, persistent object-oriented programming languages, UIDS, and associated CASE tools) to support database application development have given hope that the burden of developing DOODBMS application could be substantially reduced.

The purpose of this chapter is to propose a guideline that describes the steps taken in the phases of the development and implementation for a DOODBMS application. This guideline is based upon the previous chapter (Chapter 2) that describes various practices and methods that are instrumental in developing the application.

3.2 Summary of the Proposed Guideline

The guideline suggested for a DOODBMS application design is summarized as follows:

3.2.1 Six Activities of DOODBMS Application Design.

1. Start with OOA. In this activity: first, perform OOA as suggested in Section 3.4 to define the boundary of the problem domain. Second, gather user access information for the distributed environments, including database information, application information and user access pattern information.

2. Perform high level design: transform the model obtained in the OOA into OOD. In this activity, perform high level design as the following steps (suggested in Section 3.5).

   (a) Treat the results of the above steps as conceptual design, put the results of the conceptual design into global conceptual schema of the application database design. Combine Kim's(85) "Six Easy Steps of Object-Oriented Database Schema Design" to refine the global schema design.

(b) Perform Distribution Design: partition objects according to system responsibilities and user view. In this phase, also consider the network partition, especially network isolation problems in the distributed environment.

(c) Perform Object placement: place objects according to system responsibilities, cost and performance considerations. Consider the trade-offs of putting the objects into the distributed global database in three ways: 1) declare shared attributes, 2) create new objects, 3) create new versions. Choose an appropriate solution.

3. Low level design:

   (a) Perform Local Conceptual Schemas Design. In this activity, perform low level design as suggested in Section 3.6.

   (b) Design user interface.

4. Selecting appropriate supporting platforms and CASE tools.

   (a) Select a DOODBMS platform.

   (b) Select a UIDS.

   (c) Select CASE tools to aid analysis, design, implementation, integrating, testing, and maintenance of the application (including user interface).

5. Coding (OOP):

   (a) Apply OO methodology using OOPL by mapping pseudo code to classes, objects and structures in chosen OOPL or code generators.

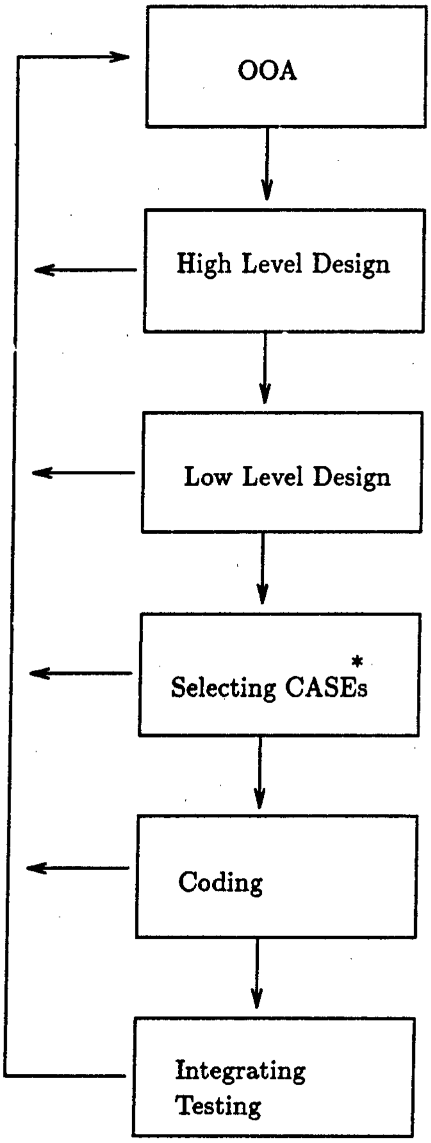
   (b) Apply the methods of structured programming as appropriate.

6. Build up the system based on the results achieved from the above steps. Observe and monitor the system to refine the application design.

Figure 3.1 illustrates this DOODBMS application design life cycle.

The following items explain more detailed of these six activities:

1. Analysis: Two concurrent sub-activities taken:



Perform requirements analysis
Gather user requirements
specifications and software
requirements specifications.

Perform OOD conceptual design
global schema design,
view design
distribution design, object placement.

Determine object representations
Refine inheritance strucutre.
Implement Object methods.
etc..

Selecting DOODBMS platform
Selecting associated CASE tools

Applying OOP

Monitoring and modifying

(* : CASE tools aid the development process. Therefore, CASE tool selection should be carried out right after the developer has understood or increasingly understands the problem domain of the targeted system using OOA and high level design technique.)

Figure 3.1. DOODBMS Application Design Lifecycle

- OOA:

    (a) Identify Classes and Objects through ADTs: identify the key abstractions of the database.

    (b) Identify the structure of these objects and classes.

    (c) Identify object attributes: define the characteristics of each object.

    (d) Identify the subject for each abstraction.

    (e) Identify the services between objects: define methods that specify what functions each object will need and can provide.

    (f) Define a data dictionary for objects&classes, services, and attributes.

    (g) Test the design of these classes and objects.

    (h) Apply inheritance where appropriate.

- Gather User Access Information:

    (a) Gather database information.

    (b) Gather application information.

    (c) Gather user access pattern information.

2. High level design: transform the model obtained in the OOA into OOD.

- OOD

    (a) System Design: set trade-offs priorities. Design trade-offs should be made in the process of development of software.

    (b) Identify superclasses, subclasses: to achieve a good data schema modeling by class refinement and reusing the codes.

    (c) Resolve multiple relationships: specify how *many-to-many* and *one-to-many* relationships are represented in the database.

    (d) Apply inheritance as appropriate.

    (e) Refine generalization, specialization, aggregation, and association: to better modeling the data schema.

    (f) Design the Problem Domain Component (PDC).

(g) Design the Human Interaction Component (HIC).

(h) Design the Task Management Component (TMC).

(i) Design the Data Management Component (DMC).

- Global Conceptual schema Design:

  (a) Put the results of the above two activities into Global Conceptual schema of the application database design.

  (b) Combine Kim's(85) "Six Easy Steps of Object-Oriented Database Schema Design" to refine the global schema design.

- Distribution Design:

  (a) Partition objects according to system responsibilities and user view.

  (b) Consider the network partition.

- Object placement:

  (a) Place object according to system responsibilities and cost and performance considerations.

  (b) Consider the trade-offs of putting the objects into the distributed global database. Choose a appropriate solution.

- Local Conceptual Schemas Design:

  (a) Treat each entity (object) as a unit of distribution.

  (b) Perform low level design as suggested in chapter 3 (the section on low level design).

3. Low level Design: continue design activities into the coding of the system.

   (a) Object representation: decide how to implement the objects: a class, data type, or static object.

   (b) enhance inheritance structure.

   (c) Implement object methods: implement methods with respect to the design strategy.

(d) Establish object visibility: ensure necessary data paths exist and explicitly hide information within each object.

(e) Identify Polymorphic Methods: it is primary an issue of implementation. With the aids of OOPL, the use of polymorphism is simply another way in which to implement a method.

(f) Perform physical design: maps the local conceptual schemas to the physical storage devices available at the corresponding sites.

(g) Design user interface:

  i. prototype a user interface to allow user to play with.

  ii. refine the user interface to satisfy the users.

4. Select Appropriate Platforms and CASE tools:

   - Select a DOODBMS platform:

     (a) Perform a survey of commercially available DOODBMS platforms .

     (b) Select a platform that is suitable to the need of the organization and the application.

   - Select a UIDS:

     (a) Perform a survey of commercially available UIDSs.

     (b) Select a UIDS that is suitable to the need of the organization and the application.

   - Select CASE tools to aid analysis, design, implementation, integrating, testing, and maintenance of the application (including user interface):

     (a) Perform a survey of commercially available CASE tools.

     (b) Select CASE tool(s) based on some certain criteria that are suitable to the need of the organization and the application.

5. Coding (OOP):

   (a) Apply OO methodologies and principles using OOPL by mapping pseudo code to classes, objects and structures in chosen OOPL or code generators.

   (b) Apply the methods of structured programming as appropriate.

6. Build up the system based on the results achieved from the above steps.

   (a) Build up the application.

   (b) Observe and monitor the performance of the application and refine.

   (c) Improve the suitability of the user views.

3.3 Feasibility Study

Before performing analysis and design for an application, the feasibility issues should be addressed first. A feasibility study is carried out to determine the need for a new information system and whether it makes sense to develop one from scratch or to adapt an existing system. Information model requirements and architectural recommendations are two inputs to the feasibility study. Information Model describes the information requirements for the user group. This information includes(8):

- Who will be using the application?
- Availability requirements from the users.
- Performance requirements
- What operations each type of user will be performing?

Architectural recommendations are to describe the network topology, location of data servers and workstations, hardware and software technologies, user access, security issues, and related design issues(8). The study should include a cost-benefit analysis to allow management assess the merits of developing a new system. The current technology support and personnel shortfalls are considered.

One way of approaching a targeted system is to consider the following questions and solutions:

- Ask "Why?" to the will-be targeted system: The first approach to a will-be targeted system is to ask the question "Why?". "Why target that system?" Analyze the advantages and disadvantages of the system, then adopt the preferred choice. Suppose that system is targeted, the advantages are the objectives and the disadvantages are to be avoided as much as possible.

- Ask "Who?", " Where?", and "When?" to better manage resources, risk management(21), and current technology support: The second approach is to ask "Who?", "Where?", and "When?". Personnel shortfalls, unrealistic schedules and budgets, and straining computer-science capability are among the top ten software risk items(21). To tackle these problems, some risk management techniques are suggested(21): 1) Staffing with top talent, job-matching personnel and do cross training. 2) Performing detailed multi-source cost and schedule estimation, do incremental design, and reuse software component. 3) Performing technical analysis, cost-benefit analysis, prototyping or use Spiral Model(20) to reduce the risk, thus increase the feasibility.

Supposed the high-level managerial decisions are made to support, then "What" and "How" can be addressed. As for "What" and "How", they theoretically correspond to analysis, design and implementation respectively, which are addressed in the latter sections.

3.4 Analysis

The analysis process used in this thesis project was based mainly upon the methodology advocated by Peter Coad and Edward Yourdon(34) with two steps added, namely, define data dictionary, verify and validate. The reasons that this method is chosen are as follows: 1) it is systematic and consistent , 2) it is simple and widely applicable, 3) it is a natural way to transfer the results of OOA to OOD to OOPL, 4) it assumes only *inheritance* about the analysis will be implemented, 5) it is supported by automated tool (i.e., OOA Tool), 6) easy to learn (less diagram notations, use familiar notations (such as Flow chart), 7) required less background to learn (assume only knowledge of OOPL, while others (i.e., OMT) require knowledge of structure analysis, 8) Coad and Yourdon have mapped the proposed OOA into the DoD-STD-2167A (Defense System Software Development), 9) it is a clear, well-organized methodology, complete with practical "how to" sections supporting each identified activity, 10) it blends data-oriented information modeling approach and entity-relationship approach, 11) define data dictionary for documentation, and 12) test the analysis by verifying and validating the classes and objects.

Also gathering user access information is preferred to be performed as part of the requirement analysis because the information obtained in this activity may affect the design(141). These specific steps of analysis are:

- OOA:

  1. Identify Classes and Objects: identify the key abstractions of the database.
  2. Identify the structure of these objects and classes.
  3. Identify object attributes: define the characteristics of each object.
  4. Identify the subjects of these classes and objects: find the relationship between the classes in the application database.
  5. Identify the services between objects: define Methods that specify what functions each object will need and can provide.
  6. Define a data dictionary for objects & classes, services, and attributes.
  7. Test the design of these classes and objects.
  8. Apply inheritance where appropriate.

- Gather User Access Information:

  1. Gather database information.
  2. Gather application information.
  3. Gather user access pattern information.

  Figure 3.2 illustrates the proposed sub-activities to take in analysis.

- Identify Classes and Objects: Coad and Yourdon(34) suggest that pertinent class-and-objects within a problem domain and within the context of a system's responsibilities are not just there for picking. We need to apply specific points of strategy to achieve good analysis. The following strategies proposed by Coad and Yourdon(34) have been applied to the thesis application analysis.

  – Strategy of Finding Classes-and-Objects:

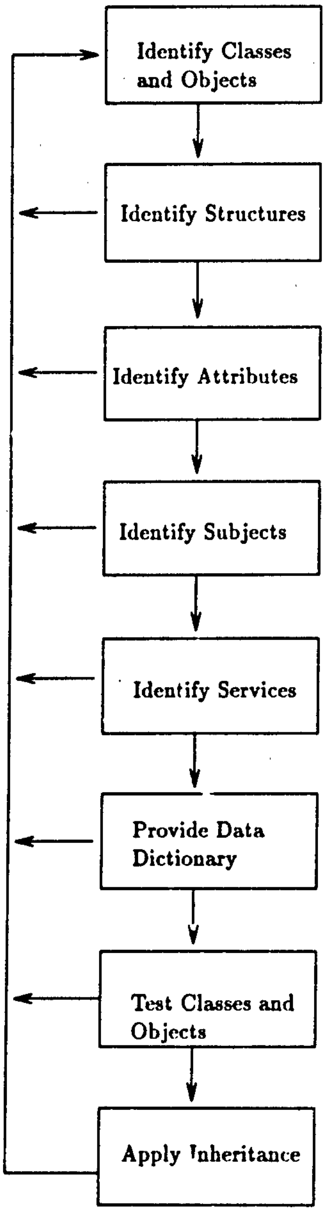


Figure 3.2. Activities of DOODBMS Application Analysis

1. How to name: Singular noun or adjective and noun; describe a single object in the class; standard vocabulary for the problem domain.
2. Where to look: Investigate the problem domain: observe first hand; check previous OOA results; read, read, read; and prototype.
3. What to look for: Look for structure, roles played, operational procedures, things or events remembered, and organizational units.
4. What to consider and challenge: Needed remembrance, needed behavior, multiple attributes, more than one object in class etc..

- Identify the Structure of Classes and Objects: As defined by Coad and Yourdon(34), the term "Structure" in OOA is defined reflecting both the problem domain and the system's responsibilities.

  - Strategy for identifying structure:

    1. Identify Gen-Spec structure: Hierarchy and Lattice structure
       (a) Is it in the problem domain?
       (b) Is it within the system's responsibilities?
       (c) Will there be inheritance?
    2. Identify Whole-Part structure:
       * What to look for
       (a) Assembly-Parts
       (b) Container-Contents
       (c) Collection-Members(and its diTerent varieties)
       (d) What to consider and challenge?
    3. Identify multiple structure

- Identify the Attributes Among Classes and Objects: Attributes add detail to the "Class-&-Object" and "Structure" abstractions.

  - Strategy for Defining Attributes:

    1. Identify Attributes.

2. Apply inheritance in Gen-Spec structures.
3. Map instance connection.
4. Check special cases.
5. Specify the attributes.

- Identify the Subjects Among Classes and Objects: In modeling the problem domain using OOATool, abstract out the details to a subject entity. This is very useful when system has strong inheritance structure. Because the subject layer describes a very high level generic representation of OOA results which can be inherited to subclasses. The shortage is that this subject layer representation does not explain very well when system with many Whole-Part relationships because the behaviors of "Whole" class do not reflect the "Part" class to much detail(54).

  - Strategy for Identifying Subjects:

    1. How to select: Promote the uppermost Class in each structure upwards to a subject. Then, promote each Class-and-object in a structure upwards to a subject. Check previous OOA results.
    2. How to refine: Refine subject by using problem sub-domain.
    3. How to construct: On the subject layer, draw each subject as a simple rectangular box, with a subject name and number inside.

- Identify the Services for Each Abstraction: A Service is a specific behavior that an Object is responsible for exhibiting.

  The essential issue in defining services is to define required behaviors(34). These services could be algorithmically-simple (such as Create, Connect, Access, and Release) or algorithmically-complex (such as calculate and monitor). Most of the Services that has been defined in the application database are algorithmically-simple, since the operations are limited to retrieve, create, and modify from the database.

  - Strategy for Defining Services:

    1. Identify object states.
    2. Identify the required services

3. Identify message connections.

4. Specify the services.

The operations are the methods or procedures for each class(132). This sub-activity defines the messages that objects send to each other. Messages help the design team to communicate and can be used to write scenarios.

- Implement Classes and Objects: There are several commercially available platforms to help prototyping OODBMS applications. ITASCA is selected to prototype the Application. The C/C++ Interface of ITASCA provides an OOPL approach that allows designers implement Classes and Object as in C++.

- Provide a Data Dictionary: Data dictionaries are repositories in which to store information about all data items (objects, services, attributes) defined in OOA. Several approaches have been proposed: 1) write a paragraph precisely describing each object class and describe the scope of the class within the current problem and define the associations, attributes and operations(144), 2) describe the content of information items(132). Three sub-activities(53) were selected to define data dictionary: 1) define objects&classes, 2) define services, 3) define attributes. For each of the data dictionary should briefly describes the following contents(53):

  1. Responsibility: define the system responsibility.

  2. What is it?

  3. What does it do?

  4. What objects does it interact with?

  The data dictionary should be expanded until all composite data items have been represented in terms that would be well known and unambiguous to all readers.

- Test the Design with Scenarios: One shortage of Coad and Yourdon's OOA methodology is that verification and validation (V&V)(53). Testing the system during requirement analysis is very helpful to insure the right product. One good way is to test the design with scenarios. Scenarios, consisting of messages to objects, test the design's ability to match the system's requirements(132).

- Apply Inheritance Where Appropriate: One of the OOP noted features is the reusability of source code. After identifying the data abstractions for each subsystem, try to introduce inheritance as much as possible.
- Gather User Access Information: Information required for distribution design includes the following:
  - Database Information: how the classes and objects are connected to each other. In the Coad and Yourdon OOA, object connections are depicted in the message connection and instance connection.
  - Application Information: Both qualitative and quantitative information are required about application to guide the distribution design(121). The qualitative information is to guide the distribution design activities, whereas quantitative information is incorporated into object placement. It has been suggested that: the most active 20 percent of user queries account for 80 percent of the total data access(172). This "80 /20 rule" may be used as a guideline in gathering the information.

3.5 High Level Design

When the initial analysis complete, the design phase began. The purpose of this phase is to bring the analysis into design. In object-oriented analysis, entities, relationships, concepts etc. were mapped into design components. In object-oriented programming, implement each design component using whatever facilities the programming language provides.

As Booch(23) suggested, object-oriented design is an incremental and interactive Process. Because most software systems are unique, and requirements may change during development, the best approach to design process is to perform a stepwise design fashion based upon new understanding or requirements.

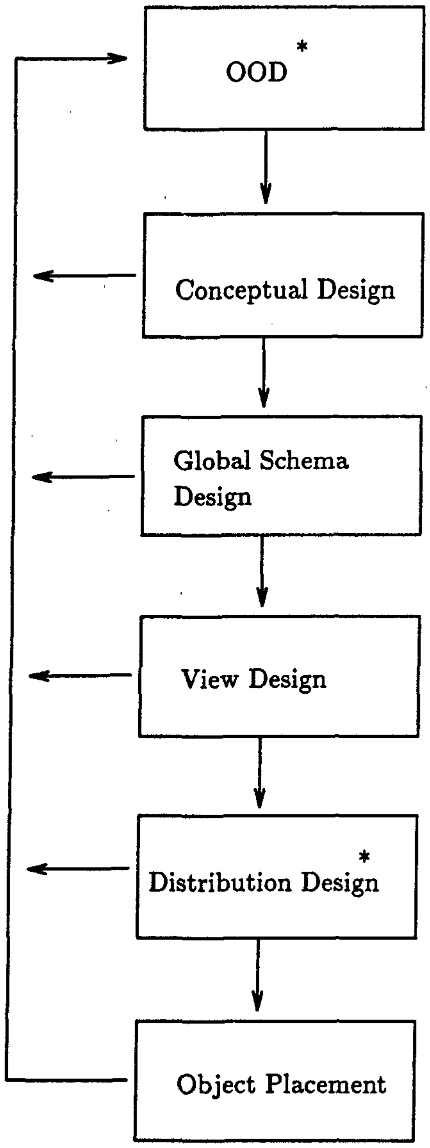
The sub-activities taken in designing the application database are combinations of object design approach from Rumbaugh et al(144) and OOD design methodology from Coad and Yourdon(35). The reasons of deriving a combined methodology is as follows: 1) provide a detailed basis for low level design, 2) facilitate the results of OOA into OOD, 3) set system design priority, 4) provide a global database design, 5) to find a suitable approach to design DOODBMS applications.

1. OOD:
   (a) System Design: set trade-offs priorities. Design trade-offs should be made during development of software.
   (b) Identify superclasses, subclasses: to achieve a good data schema modeling by class refinement and reusing codes.
   (c) Resolve multiple relationships: specify how *many-to-many* and *one-to-many* relationships are represented in the database.
   (d) Apply inheritance as appropriate.
   (e) Identify generaliz.ation, specialization, aggregation, and association: to better modeling the data schema.
   (f) Design the Problem Domain Component (PDC).
   (g) Design the Human Interaction Component (HIC).
   (h) Design the Task Management Component (TMC).
   (i) Design the Data Management Component (DMC).
2. Conceptual DOODBMS Design.
3. Global Schema Design.
4. View Design.
5. Distribution Design.
6. Local Schema Design.
7. Object Placement.

Figure 3.3 illustrates these sub-activities. In this Figure, the designers should notice that the activity of "Distribution Design" will change the results obtained in the activity of "OOD". In these activities, the decision of replication alternatives (refer to Table 2.11

for a replication alternatives' trade-offs) will affect the results especially when partial ana pure replications are adopted. One simple solution is to use dynamic schema evolution facilities provided by the DBMS platforms to modify schema dynamically (refer to Table 2.2 and Table 2.1 of Chapter 2 for a comparison of some major OODBMSs and DDBMSs in this facility).

- OOD: In this activity, the following sub-activities are taken:
  - System Design (Set Trade-offs Priorities): The first sub-activity of the high level design process was to settle upon a design strategy for the construction of the design. The step is derived from the step 8 of Rumbaugh et al's "system design": "Setting trade-offs priorities"(144). In this step, decision is made as to what gets priority during the development. Decide between such factors as performance, memory space, portability, cost, maintainability, and understandability. During the process of design, trade-offs priorities are set to guide the design process. The decision was made to give the performance as the highest priority. As the DBMS platform maintains its own integrity on data, it usually incurs performance overhead. The application should be given the speed gets the priority. The second priority was made to the maintainability and re-usability.
  - Identify Superclasses, Subclasses: The purpose of this sub-activity is to achieve class refinement and reusing the codes. In class refinement, the designers examine each class in the results obtained from the analyses phase with respect to reusability. Because the results from analysis lacked methods and attributes to make them more reusable in other domain. The methods and attributes needed to enhance this function are added in this step. This step is similar to (144) step 5 in "Steps of Object Design". During the design process, the classes and operations can often be adjusted to increase the amount of inheritance. Refining the class structure, such as abstracting out common behavior or rearranging classes and operations, can increase the amount of inheritance.
  - Apply Inheritance as Appropriate: The goal of inheritance refinement is to further organize the design and to take advantage of the commonality between



(* : The designers should notice that the activity of "Distribution Design" will change the results obtained in the activity of "OOD".)

Figure 3.3. Activities of ΓOODBMS Applicntion High Level Design

classes. While defining classes and objects, new classes could be created where they can be used by subclasses. Once commonality had been identified, standardization of protocols come next. This technique is achieved by defining additional "root" superclass that contained the description of the attributes and methods that should be provided by all subclasses derived from them. Besides, inheritance makes maintenance easier in this application, which use many simple class lattice structures. In this case, inheritance reduces the likelihood of human error because changes in one class is automatically propagated to all subordinate classes. Standardization of protocols aids reusability by defining a default interface for all the different variations of the root classes. The last step in inheritance refinement is to identify abstract classes. In factoring the inheritance structures, there are classes that served only as parts of others. The abstract classes existed only to provide pieces for derived classes.

- Refine Generalization, Specialization, Aggregation, and Association: Gen-Spec and Whole-Part structures are desired features in the problem domain. Keep asking the following questions as suggested by Coad and Yourdon(34): "Is it in the problem domain?" "Is it within the system's responsibilities?" "Will there be inheritance?" Also, consider each Class as a specialization. For its potential generalizations, ask similar questions.

- Resolve Multiple Relationship: The purpose of this sub-activity is to identify the strategy that will be used to implement multiple relationships. Part of the analysis revolved around defining the multiplicity of the relationship between classes and objects that participated in a composite relationship. This step allows designers to decide how many-to-one and many-to-many relationships will be implemented.

- Design Coad and Yourdon's OOD Four Component: Coad and Yourdon propose that the results of OOA can be put right into the PDC. The idea is to use the results of OOA and add to them within the constraints of Coad and Yourdon method. These additions do not mean it is time to hack up analysis results into design. Instead, several criteria are proposed(34), (35) to use when adding the OOA results during the construction of the PDC:

- Reuse design and programming classes: look for existing classes to reuse.
- Group problem domain specific classes together.
- Establish a protocol by adding a generalization of a class: add a class to formalize the inheritance of derived classes.
- Accommodate the supported level of inheritance: it may be necessary to revise the various inheritance relationships if multiple inheritance of any form on inheritance is not supported.
- Improve performance: may need to rearrange or combine classes to reduce message traffic.
- Support the Data Management Component: to support the Data Management Component, each object is store or sent to another object designed to saved objects.
- Add lower-level components: to be more convenient and understandable.
- Don't modify just to reflect team assignments: do not separate related classes between different software development teams.
- Review and challenge the additions to OOA results.

The PDC is a key to keeping the stability of the four domain based throughout the analysis, design, implementation, and reusability. Any modification to the PDC is a trade-off between expressing the modification criteria and maintaining the stable representation of the problem domain. With the object-oriented paradigm, one is highly motivated to keep the problem domain organizational framework intact. Because modifications consequently affect the potential reusability of the model with other systems of similar problem domain. The results of PDC will likely change due to changing requirements, technology (such as inheritance capability of the implementation language), lack of understanding during OOA, etc..

The HIC captures how the people who use the system and the user interface that is required for them to take advantage of the maximum system capabilities. The main concern is: classify organizational and user skill level and design a suitable user interface to satisfy the user's emotional, organizational culture without sacrificing the functionality of the application. The TMC is to decide to task or not to task. Tasking decisions involve identifying eventdriven tasks, clockdriven tasks, coordinator, and establishing priority and critical tasks. Tasking adds complexity to design, coding, testing, and maintenance. It is important to keep task to a minimum and coordinate then well. The DMC is to provide the database management system infrastructure by isolating the impact of data management scheme, whether flat file, relational, or object-oriented data models. Coad and Yourdon propose guideline for designing the data layout of flat files, relational, and object-oriented (both extended relational approach and extended OOPL approach). For a clear look and understanding of the methodology, refer to (34), (35).

- OO Tool: ObjectTool[1] combines OOA and OOD methods that are proposed by Coad and Yourdon can be used to help design the application database. The reason is consistent with the OOD methodology that Coad and Yourdon Proposed. Some other OO CASE tools(37) (such as OMT, OOWorkbench, ObjectPlus, Adagen, and Object-Oriented Environment) are also commercially available.

- Conceptual Design: The purpose of conceptual design is to decide classes and objects types and relationships among these classes and objects. This process can be divided into two activities: class and object analysis and functional analysis. Class and object analysis is concerned with determining the classes and objects, their attributes, and the relationships among them. Functional analysis is concerned with determining the fundamental functions with which

[1] This tool will be marketed by Object International Inc. (Coad and Yourdon) the estimate release date will be in June 1993.

the modeled enterprise is involved. These two activities are basically the same as doing the above two activities (OOA and high level design). In OOA and high level design, objects, class, attributes, and services are identified by analyzing the user's requirements. OOA notations are graphical representations of user's view of the application. The results of conceptual design are then used in the next activity, global schema design.

- Global Schema Design: The purpose of this activity is to design the global conceptual schema by gathering the results achieved from the conceptual design. The global database design proceeds in the same manner as in the centralized database design process. Requirements are formulated, the conceptual schema is designed and the logical database structure is created. Object-oriented data model is selected for describing the global schema. The network nodes may have different user view of the data. The views are modeled and integrated until the global schema is achieved. In this phase of development, OOA and OOD have been applied to achieve a global schema design. Kim(85) proposed six "easy" steps of object-oriented database schema design. This iterative process has been adopted due to its comprehension of object orientation and easy to apply. Figure 3.4 illustrates this iterative process.

- View Design: There is relationship between the conceptual design and the view design: the conceptual design can be interpreted as being the integration of user views(121). View integration is used to ensure that classes and objects and relationship requirements for all the views are covered in this step. The purpose of view design is to deal with defining the interface for the end users. During the global database design phase, a global view of the data was constructed using the messages in the system. Each user or node of the system may not view the data in the same manner as defined in the system's responsibility and problem domain. There are two reasons for user view management: 1) the users are not necessarily interested in the entire database, 2) the users are not allowed to access the data. In relational database, the global relations of the systems are called base tables, each of which is stored as a distinct file. A *view* of the

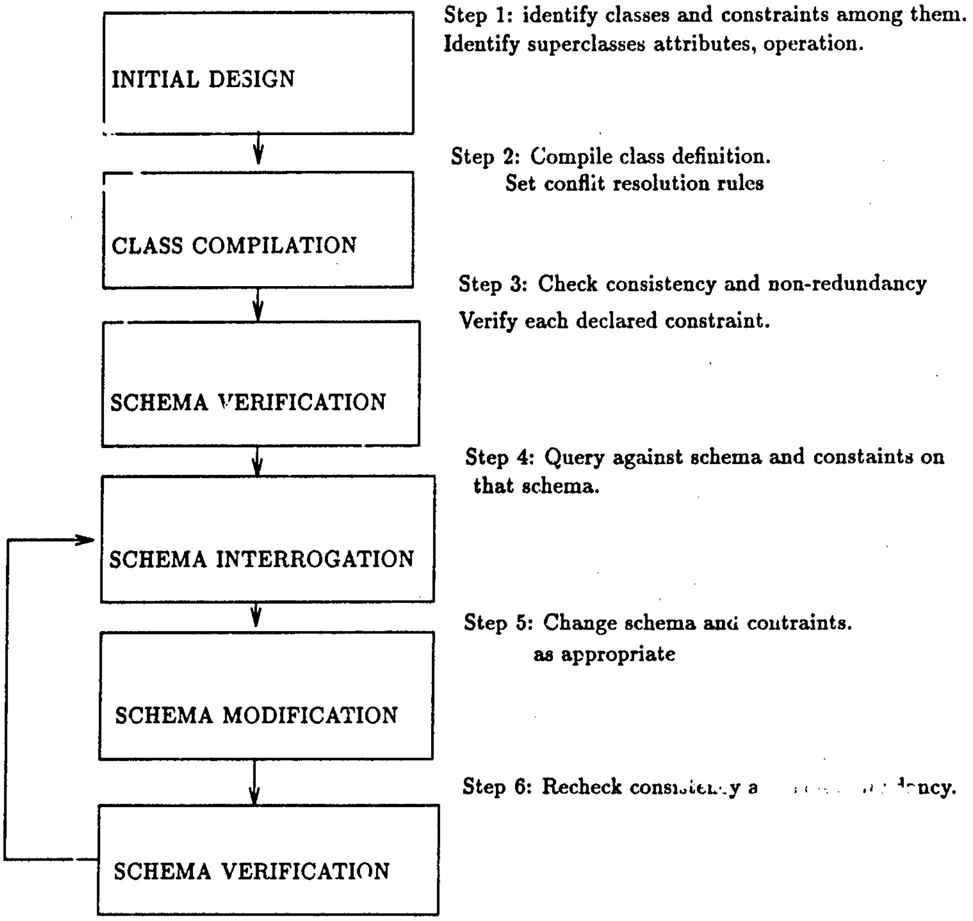

INITIAL DESIGN
CLASS COMPILATION
SCHEMA VERIFICATION
SCHEMA INTERROGATION
SCHEMA MODIFICATION
SCHEMA VERIFICATION
Step 1: identify classes and constraints among them. Identify superclasses attributes, operation.
Step 2: Compile class definition. Set conflit resolution rules
Step 3: Check consistency and non-redundancy Verify each declared constraint.
Step 4: Query against schema and constaints on that schema.
Step 5: Change schema and coutraints. as appropriate
Step 6: Recheck consistency and non-redundancy.

Figure 3.4. Kim's Six Easy Steps of Object-oriented Database Schema Design

data may be thought of as a virtual table. The same approach can be applied to create view for object-oriented databases by creating new objects as *views.* For example, in the thesis application, the object-&-class *EMPLOYEE* can be divided by creating new classes *EMPLOYEE1* , *EMPLOYEE2*, *EMPLOYEE3* with only required attributes for these derived objects based on the system responsibilities and problem domain.

The advantages of defining user *views* are: 1) allow users have the ca pability to focus on only the subset of data with which they are concerned, 2) the same data can be viewed by many users in different ways, 3) the database can be expanded or restructured in other ways without obstructing the users, and 4) data security is provided. In terms of the view design methods provided by the ITASCA system, many facilities or functions are provided: 1) set the query scope of objects to either *shared* or*private* is provided, 2) make versioned objects, then declare that versioned object is either shared or private, 3) change attributes of objects to shared or private attributes.

- Distribution Design: In RDBMS, fragmentation and allocation are two major activities in distribution design. In DOODBMS, distribution of objects and object placements are the primary concern. The distribution of objects can be achieved by: 1) declare shared attributes, 2) create new class and object, 3) create new version of class and object. The object placement is derived from the FAP models that has been proposed by Chu(32), Bray(24) and Ceri et al(30).

A simple yet practical guideline is useful. Consider the following six factors(24) for distribution design:

1. *Storage Cost:* The costs for storing the files are the unit cost times the length of the files. Bray(24) suggests that more active files should be stored on faster, yet expensive devices, while those less active files could be stored in an archival storage.
2. *Communication Cost:* A three dimensional matrix can be used to define the communication costs. The matrix, C(i, j, k), is used to specify the unit

costs of transferring file i from node j to node k. These unit costs must be multiplied by the file length to obtain the total costs.

3. *Retrieval Rate for Each File at Each Site:* The retrieval rate for each file at each site is an important factor to decide how the distribute the objects.

4. *Storage Capacity at Each Site:* The storage capacity constraints should be considered even though this problem is less concerned today.

5. *Update Rate for Each File at Each Site:* The frequency of modification after each transaction is also a main consideration. If the files are stored with highest update rate in the local computer, the designers can minimize the cost by avoiding the communication costs.

6. *Maximum Allowable Access Time at Each Node for Each File:* The maximum allowable average retrieval time should be considered. If there is a time constraint to certain local computers, then the designers should consider storing the file locally.

* General Criteria for Fragment Allocation: General criteria for fragment allocation are suggested by Ceri(30). He classified allocation as two forms:

  1. Nonredundant Allocation: Ceri suggests the simplest method is a "best-fit" approach, i.e., a measure is associated with each possible allocation, and the site with the best measure is selected.

  2. Redundant Allocation: Replication introduces further complexity in the design. Because the degree of replication of each fragment becomes a variable of the problem, and modeling read application is complicated by the fact that the files can be selected from several alternative sites(30). For determining the redundant allocation, Ceri says either of the following two methods can be used: 1) Select the highest benefit of allocating fragments to each site. This method is based on the operation frequency of each site. 2) Determine first the solution of the non-replicated problem, and then progressively introduce replicated copies starting from the most beneficial. The process is terminated when no "additional replication" is beneficial.

- * A General FAP Solution with Query Processing

    Solving combined query and allocation is extremely complex. In general, a query may access multiple files. A query processing strategy schedules the order that the files are accessed. To solve the query processing problem, Wah(168) suggests two steps:

    1. The query processing order is optimized independently for each query or update providing that distinct copies of files are used.
    2. Locate distinct copies at virtual sites and the file allocation becomes the mapping of the virtual sites to physical sites so that the total communication costs are minimum.

– Local Schema Design: The purpose of the sub-activity is to partition global schema and allocate files based on the users requirements and objectives of processing locality, availability, reliability, and storage requirements. In this phase, apply the guideline from the distribution design, treat each entity as a unit of distribution and perform object placement and low level design.

– Data Placement: One of the key aspects of distributed application is how they manage data. Given that a set of objects to share in the distributed environment, there are three choice(71): 1) Centralized data management: place the data at one node and route all accesses to the node, 2) Replication data management: make copies of data at various node; the nodes needing access the data can access the "closet" node, and 3) Partitioned data management: partition data into pieces and place the pieces at different sites; access to the data must be routed to the appropriate node depending on which piece of data is being accessed.

- * Heuristic Techniques for Object Placement: Since optimal FAP solution is complicated and difficult to achieve, seven heuristic techniques of object placements for minimizing the total network communication costs are suggested. These techniques achieved a sub-optimal file allocation solution despite the complexity of optimal placements. These techniques represented a set of extremes and a representative range of possible methods that may be used to compare costs and to assist the designers in choosing a suitable allocation of data items. For instance, the preliminary design is modeled based on the user view, system responsibility and problem domain, then the developers put the system to collect access information on query and update rate from the users, then refine the allocation decisions to achieve sub-optimal, yet, acceptable solutions. These techniques are as follows(121), (30), (34), (35):

1. Placement by Full Replication.
2. Placement by User View.
3. Placement by System Responsibility.
4. Placement by Greatest Query Rate (Redundant).
5. Placement by Greatest Query Rate (Non-redundant).
6. Placement by Least Update Rate (Redundant).
7. Placement by Least Update Rate (Non-redundant).

* Placement by Full Replication: This step is the first try-out design. The objects are replicated in every node that participated the global database. This approach is refined when the objects access information is clear to the application designers and users.

* Placement by User View: This placement technique determines the cost to store each user's view locally. Assume the local storage capacity is unconstrained. This technique decides the total network communication costs and the required memory size at each node.

* Placement by System Responsibility: This allocation technique allocates objects according system responsibility and problem domain. After the problem domain and system responsibility have been defined, the allocation followed. In this alternative, the following steps are performed: 1) Consider the system responsibility: Place the critical objects in the local sites. 2) Consider the network configuration: network partition (such as isolation)

may cause objects inaccessible, decision should be made to place those objects in the local site.

* Placement by Greatest Query Rate (Redundant): This technique uses the solution produced by the previous technique as its basis. Redundancy is introduced to decrease the total communication costs. Before applying this technique, access information should be gathered to decide the placement method in this redundant fashion.

* Placement by Greatest Query Rate (Non-redundant): Another option for allocating data is to locate each object at those nodes where the query rate for that object is the highest in a nun-redundant fashion. Such a technique should reduce the total query costs. Before applying this technique, access information should be gathered to decide the placement in this nun-redundant fashion.

* Placement by Least Update Rate (Redundant): This technique uses the solution produced by the previous technique as its basis. Redundancy of objects is introduced to decrease the total communication costs. Before applying this technique, access information should be gathered to decide the placement method in this redundant fashion to minimize the operational costs.

* Placement by Least Update Rate (Non-redundant): The update cost, for a object at a node, is determined by the sum of the update rates form other nodes. For that reason, this technique seeks to place objects to those nodes where the sum of the update rates to that object from all other nodes is the least. By doing so the total network update costs is reduced. Before applying this technique, access information should be gathered to decide the allocation method in this non-redundant fashion.

3.6 Low Level Design

The purpose of low level design is to concentrate on implementing the design given the strengths and weaknesses of the language used. Most of the low level design methods in this phase are selected from Rumbaugh et al(144). The reasons that this method is derived is based on the following: 1) provide a detailed basis for implementation, 2) facilitate the results of high level design (such as OOD) into low level design (such as OOPL), 3) provide a computer-orientation required for a practical implementation.

The sub-activities taken in this activity are summarized as follows:

1. Object representation: decide how to implement the objects: a class, data type, or static object.

2. Refine inheritance structure.

3. Implement object methods: implement methods with respect to the design strategy.

4. Establish object visibility: ensure necessary data paths exist and explicitly hide information within each object.

5. Identify Polymorphic Methods: it is primary an issue of implementation. With the aids of OOPL, the use of polymorphism is simply another way in which to implement a method.

6. Apply OOD using OOPL.

7. Perform physical design: maps the local conceptual schemas to the physical storage devices available at the corresponding sites.

Figure 3.5 illustrates the steps taken in the low level design.

- Determine Object Representation: This sub-activity is derived from Rumbaugh et al(144) (step 7: Determine object representation). As Rumbaugh et al suggested: Determine object representation is to decide whether to use primitive types (such as integer, string, etc.) or implement as an object. Classes can be defined in terms of other classes, but eventually everything should be implemented in terms of built-in primitive types, such as integers, strings, and enumerated types.

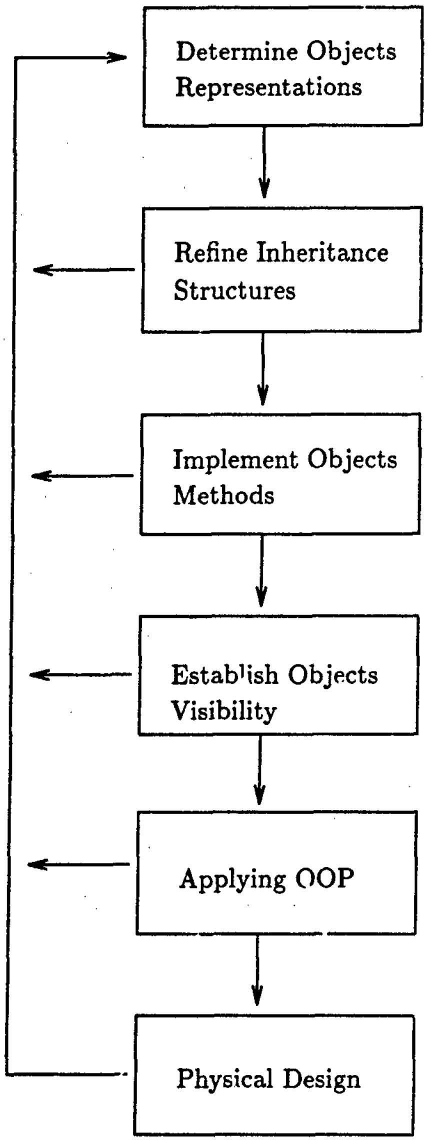

Determine Objects Representations
Refine Inheritance Structures
Implement Objects Methods
Estab'ish Objects Visibility
Applying OOP
Physical Design

Figure 3.5. Activities of DOODBMS Application Low Level Design

There are three way to implement objects in C++(154): 1) Implement the object with a class, 2) Implement the object with a data type, and 3) Implement the object as a static object. When the candidate objects provide characteristic function and data, implement the object with a class in more nature in the database application design. When the candidate objects do not provide characteristic function and data, implement the object with a data type and static object. This may be easier to use as a data type, but this will limit the object-oriented programming power and complicate future modification(154).

- Refine Inheritance Structure: This sub-activity is derived from Rumbaugh et al(144) (step 5: Adjust class structure to increase inheritance).

  During the object design process, the definitions of classes and operations can often be adjusted to increase the amount of inheritance(144). As pointed out by Rumbaugh et al(144), the designer should perform the following procedures:

  1. Rearrange and adjust classes and operations to increase inheritance.
  2. Abstract common behavior out of groups and classes.
  3. Use delegation to share behavior when inheritance is semantically invalid.

- Implement Object Methods: This sub-activity is derived from Rumbaugh et al(144) (step 2: Design algorithms to implement operations).

  Rumbaugh et al(144) suggest that the algorithm designer should perform the following procedures:

  1. Choose algorithms that minimize the cost of implementing operations.
  2. Select data structures appropriate to the algorithms.
  3. Define new internal classes and operations as necessary.
  4. Assign responsibility for operations to appropriate classes.

New attributes were added to the class during low level design. Modification of the inheritance in the program invo.. :d organizing the design to take advantage of similarities between classes, standardizing protocols and identifying abstract classes. Three steps(154) to refine inheritance structure is suggested:

1. Try to factor commonality from classes in the inheritance structure: new classes may be created where they could be used by more than one subclass or if they constituted a usable abstraction in themselves. This should be performed with one warning in mind: "Do not introduce a Gen-Spec Structure just for the sake of extracting out a common attribute"(34). For example, extracting out the *Location* of *Organization*, *Aircraft*, and *Airport* as a commonality will be an inappropriate Gen-Spec Structure.
2. Standardize the protocols: this is accomplished by defining additional superclasses that contained the description of attributes and methods that should be provided by all subclasses derived from them. Standardization of protocols aids reusability by defining a default interface for all the different variations of the superclasses.
3. Identify abstract classes: In the process of factoring the inheritance structures, there are classes that served only as parts of other classes, such as generic superclasses. This abstract classes existed only to provide pieces for derived classes. Instantiating objects of an abstract class would make no sense because they lack the necessary attributes and/or Methods to be usable.

– Establish Object Visibility: This sub-activity is derived from Rumbaugh et al(144) (step 4: Optimize access paths to data). The purpose of this step is to ensure that all data paths exist between objects that communicate and specify the visibility of each class respect to all others in the system. Adding redundant associations for efficient access is sometimes necessary. A C++ programmer has five ways to make one object visible to another(154):

  * inheritance: inheritance relationships should be used where there are indicated in the design.
  * make one object an attribute of the other (composition): should be used where there are indicated in the design, otherwise, it should be documented.
  * make a pointer to one object an attribute of the other: allows more flexibility as opposed to using the actual object as a component. Using pointers

to objects enables polymorphism, allows the Whole-Part structure (such as Container relationships) access to an object that was possibly instantiated in another object.

    * implement one object as static class: information and include the header file that define the class,

    * pass the needed object as a parameter to all methods that require the object.

- Identify Polymorphic Methods: The concept of polymorphism is handled in low level design because it is primary an issue of implementation. With the aids of OOPL, the use of polymorphism is simply another way in which to implement a method. The best candidates for polymorphic methods are those methods that are common to all classes within an inheritance structure (such as Gen-Spec Hierarchy and/or Lattice Structure), but this does not apply to Whole-Part Structure very well.

- Apply OOD Using OOPL: This sub-activity is derived from Coad and Yourdon(35) (Designing the Data Management Component).

  In the Data Management Component Design, OOPL approach has been applied to design the data layout, because the C/C++ Interface of the ITASCA platform has been used to prototype the database. This interface extended the C++ programming languages to allow users interface with the ITASCA DODBMS.

- Perform Physical Design: The purpose of this step is to map the local conceptual schemas to the physical storage devices available at the corresponding sites(121). The inputs to this process are the local conceptual schema and access pattern information of the classes and objects.

- Design user interface: This sub-activity is similar to designing the HIC proposed by Coad and Yourdon. Beside performing the methodology of HIC, three steps were added:

  1. prototype a user interface to allow user to play with. The only reliable way to generate quality interfaces is to test prototypes with users and modify the design based on their comments(110),

2. apply guidelines for designing application GUIs: Shneiderman(151) suggested guidelines of interface design as "Eig‘ . Golden Rules". Also, guidelines for designing application GUIs are provided by GUI-specific style guides such as the OPEN LOOK GUI Application Style Guidelines(163), the OSF/Motif Style Guide(120). These documents provide high-level rules and principles that help to maximize consistency across application in the respective GUIs,
3. refine the user interface to satisfy the users.

3.7 *Selecting an Appropriate DOODBMS Platform and CASE :ools*

CASE tool aids the development process from the activity of analysis, design, implementation, to the activity of integrating and testing. Therefore, CASE tool selection should be carried out right after the developer has understood or increasingly understands the problem domain of the targeted system using OOA and high level design technique.

- Select an Appropriate DOODBMS Platform: In the process of DOODBMS application design, developers will change the schema frequently to satisfy the need of object distribution and placement. Place the results of OOA and high level design into the global schema of the distributed environment is a rough step. During distribution design, the schema may change frequently. With the aids of the dynamic schema evolution facilities provided by some DOODBMS platforms (such as the ITASCA). This problem is not difficult to solve. In this phase, perform the following steps:

  1. Perform a survey of commercially available DOODBMS platforms .
  2. Selected a platform that is suitable to the need of the organization and the application.

  Guideline of selecting suitable DOODBMS platforms: This guideline is modified from the "Seven criteria(108) of evaluating CASE tools" with added DOODBMS platform features and asked questions. The criteria of selecting

DOODBMS platforms are suggested for application developers when evaluating and selecting DOODBMS platforms as follows:

1. Power: In this criteria, ask the following questions:
   (a) Does the platform fully support distribution?
   (b) Does the platform fully support object-oriented concepts?
   (c) How fast he performance and response?
   (d) How long does it take to start up the platform?
2. Ease of use: In this criteria, ask the following questions:
   (a) Does the platform have a good user interface and can the user interface be tailored to fit different user's "idiosyncrasies"?
   (b) How about the learning curve?
3. Robustness: In this criteria, ask the following questions:
   (a) Does the platform compatible between versions?
   (b) Does the platform reasonably "bug-free"?
4. Functionality: In this criteria, ask and consider the following questions:
   (a) Does the platform support the adopted methodology well?
   (b) Consider important functionalities of DOODBMS platform (such as: dynamic schema evolution facilities, external language interface, multimedia support, multiple inheritance, lock management, change notification etc.), then compare to the need of the organization and application to select appropriate platform(s).
5. Ease of insertion: In this criteria, ask the following questions:
   (a) Does the platform contains precise and clear installation procedures?
   (b) Is the platform available on the certain hardware platforms that fit the need of the organization?
6. Quality of support: In this criteria, ask the following questions:
   (a) Is the maintenance support provided?
   (b) Is hotline service provided?

7. [illegible]s: In this criteria, consider the following conditions: user's technique, platform future development promise, cost, complies with X11/Motif standard, standard network protocol, reputation of the platform, portability, policies, etc..

- Selecting an UIDS:

1. Perform a survey of commercially available UIDSs. For a survey (including sources and vendor's information) of commercially available user interface toolkits and UIDS, refer to (110).
2. Select a UIDS that is suitable to the need of the organization and the application.

Guideline of selecting UIDS(110): This UIDS should perform the following:

1. handle the mouse and other input devices,
2. validate user input,
3. handle user errors,
4. process user-specified aborts and undos,
5. provide appropriate feedback to show the input has been received,
6. provide help and prompts,
7. update the display when application data changes,
8. notify the application when the user updates application data,
9. handle field scrolling and editing,
10. insulate the application from screen-management functions,
11. automatically evaluate the interface and propose improvements,
12. provide information to help the designer evaluate the interface, and
13. let the user customize the interface.

A number of commercially available software tools provide a GUI interface to the GUI design task. GUI builders such as Devguide(162), Interface Builder (116), and Interface Architect (62) provide the designers with standard user interface components that can be dragged onto the work surface and arranged using direct manipulation by generating source code or executables for a particular configuration of GUI components.

- Selecting Appropriate CASE tools: Select CASE tocls to aid analysis, design, implementation, int^grating, testing, and maintenance of the application (including user interface):

  1. Perform a survey of commercially available CASE tools.
  2. Selected CASE tool(s) based on some certain criteria that are suitable to the need of the organization and the application.

  Seven criteria(108) of evaluating CASE tools is suggested for application developers when evaluating and selecting CASE tools :

  1. Ease of use: In this criteria, ask the following questions: Does the tool have a good user interface and can the user interface be tailored to fit different user's "idiosyncrasies"?
  2. Power: In this criteria, ask the following: How fast is the performance and response? How long does it take to start up the tool?
  3. Robustness: In this criteria, ask the following questions: Does the tool compatible between versions? Does the tool reasonably "bug-free"?
  4. Functionality: In this criteria, ask the following questions: Does the tool support the methodology well?
  5. Ease of insertion: In this criteria, ask the following questions: Does the tool have precise and clear installation procedures? Is the tool available on the selected hardware platforms?
  6. Quality of support: In this criteria, ask the following questions: Is the maintenance support provided? Is hotline service provided?
  7. Others: In this criteria, consider the following conditions: user's technique, cost, complies with X11/Motif standard, reputation of the CASE tool, portability, policies, etc..

For a official evaluation, use certain form to keep track of the evaluation results is desired. Various practical forms (such as forms of recording tool selection criteria, forms of classifying tools, forms of tool-to-organization interconnection profile, forms for evaluating and selecting appropriate tools, and form for creating tool-interconnection profile) suggested by Poston et al (131) help evaluating and selecting tools into a more measurable task.

3.8 Coding (OOP)

In this activity, perform the following two steps:

1. Apply OO methodologies and principles using OOPL by mapping pseudo code to classes, objects and structures in chosen OOPL (refer to Table 2.8 for a comparison of some major OOPLs to facilitate the selection process) or code generators.
2. Apply the methods of structured programming as appropriate.

3.9 Observing and Monitoring the Application

In this activity, perform the following steps:

1. Build up th : application based on the results achieved from above steps.
2. Observe and monitor the performance of the application and refine.
3. Improve the suitability of the user views.

3.10 Summary

This chapter summarizes the steps to take in the phases of the development of a DOODBMS application. Before performing a development, a feasibility study is necessary to better manage resources and reduce risks. Once commitment is obtained, the development process begins. In the development process, OO does unify databases and applications(87), (44). In this unified process, OOA has been used to understand the problem domain and system responsibilities. OOD has been applied to transform the model obtained in the OOA to data schema for design. Also, the features of OOP have been applied to develop the database. In distributed database design, the global design is the steps in object-oriented database design. The local schema design addresses the issues of FAP optimal solutions that are desired but not always feasible to achieve in the application design. A sub-optimal heuristic approach was selected. This approach places objects based upon: 1) user's view, 2) system responsibility and problem domain, 3) greatest access rate, and 4) least update rate to help minimize the total operational costs. Since the development process is complicated and expensive, tools for integrating computer-aided software engineering are desired to increase productivity and quality. But CASE tools adoption is not optimistic. Organizations tend to adopt integrated tools only in a limited fashion. Part of the answer may lie in a misinterpretation of the learning curve and its affect on productivity, and part of the problem is that no one agrees on how the learning curve is likely to affect tool adoption(77).

The object-oriented paradigm is still a new technology. Likewise, DOODBMS is complex and not yet well-understood. The guideline proposed does not mean that the steps should be followed in a rigid manner since the object-oriented develop process is a "round-trip gestalt"(23) process. It is developer's choice to apply this guideline or to improve it for developing DOODBMS applications. Meanwhile, the concept of "unlimited formalization"(41) for not imposing formalization upon the analysts, especially in Object-Oriented technology (diverging modeling techniques), is recommended. Because a developer should be able to be as informal as necessary for a particular task. Thus a method or guideline should not put a limit on achievable formality. Consequently its basic concepts should be crystal clear, unambiguous and have preferably a formal semantics.

IV. Requirements Analysis and Preliminary Design

4.1 Introduction

The objective of this chapter is to achieve a requirement analysis and preliminary design for the targeted database application by applying the methodology in the previous chapter.

4.2 Requirements Analysis

The boundary of the problem domain of the targeted system can be defined by performing OOA. This distributed DBMS application maintains information as listed in Appendix E. OOA, OOD results are located in Appendix C and Appendix D respectively.

4.3 Design Considerations of the Application

One major concern of DOODBMS design is how to place objects (data). Data placement is a function of the following parameters(142):

1. Level of sharing: The distributed database can be perceived as one global database. If the data is private, keep the data at the workstations. The user in the local site participates the distributed database can either put the data in the global database or declare shared data to various role of users.
2. Update/Access frequency: According to Rofrano(142), to get the shared data to a corporate level, the following questions should be addressed:
   (a) How often are the data accessed?
   (b) How much is required?
   (c) How often are the data updated?
   (d) How often is the shared data to have the latest copy?
3. Security: The geographical distribution of nodes of a distributed system introduces security risks that do not exist in a centralized system. Approaches to

solving these problems are being developed, such as better password mechanisms, data encryption for stored data and messages, etc.

4. Capacity: Capacity is becoming less of an issue these days. If the developer determines the best placement for data, only to find that there is not enough capacity for the data on that platform, then the developer should either increase the capacity on that platform (i.e., workstation) or move the data to the next "up-stream" platform (i.e., file server) that has the capacity.

5. Organization Needs: The policy of an organization may have different needs to place data (either decentralized or centralized). Whatever the need, it should be balanced with what is available with current technology to satisfy the local data requirements. Trade-offs should be made as to where to place the application function given the current data placement.

Designing an effective DOODBMS is a difficult task, because there are many competing requirements. The functional requirements of the application are subject to time and space factors constraints. To achieve time-efficiency and cost-effectiveness of our application database, object-oriented analysis should be performed carefully. Object-oriented analysis seeks to identify the classes and objects that are common to the application.

OODBMSs provide several mechanisms to model different relationships among objects. Object specialization/generalization (Gen-Spec) refers to the ability to organize objects in an IS-A (Whole-Part) relationship, while classification is the ability to relate an object to a group of objects via the INSTANCE-OF relationship. Also aggregation allows the designer to model an object as its constituent objects. This type of relationship is known as the IS-PART-OF relationship. In OOA, we build requirement models that expose the existence and behavior of entities, entity types, relationships, and relationship types, along with events in the user's world that affect the design. We also need to be sure of the allowable states the entities can assume and the allowable transitions between those states. If DBMSs are to replace the flat files management systems that are common in engineering design applications, the performance of these DBMSs should be comparable to the existing systems. The following sections discuss methods of evaluating database performance and some design issues that may affect the performance of DOODBMSs.

- Performance Considerations: Much of the literature on database performance evaluation addresses the results of standard benchmarks as applied to various DBMSs. While most of these benchmarks reflect typical applications for relational DBMSs. Cattell has developed an approach for measuring the performance of OODBMSs(29). He summarizes three most important measures of performance, based on an earlier benchmark for simple database operations, in an OODBMS:

    * Lookup and Retrieval: Look up and retrieve an object given its identifier.
    * Traversal: Find all objects in the hierarchy of a selected object.
    * Insert: Insert objects and their relationships to other objects.

    Cattell suggested that OODBMS should improve performance by a factor of ten to 100 times than that of traditional DBMSs to be adopted for engineering applications(29). This is reasonable because engineering applications are very complex and difficult to handle efficiently. Martin and Odell(103) suggests that the performance of OODBMS is improved over the traditional DBMS (RDBMS) when the following measures are taken:

    * The use of software pointers to point to another object. Traditional DBMS must use join to access other tables (slower).
    * OODBMS makes physical clustering more effective. RDBMS spreads data into multiple tables.
    * OODBMSs use diverse storage structure, such as multimedia support and CAD support. Multimedia applications required storing large data streams representing digitized audio and video data. CAD applications require storing large numbers of very small objects. Neither is suited for RDBMS as an efficient purpose.
    * avoid redundancy: OODBMS use inheritance to reduce redundancy of data and methods.

In OODBMS, an attribute can be a simple type, simple class, a set of simple type, or a set of classes. The use of software pointers increases the performance of database access to the same physical clusters of media as the redundancy is avoid by using inheritance as much as possible.

4.4 Applying OOA to Analyze the Application

The OOA strategy proposed by Coad and Yourdon has been applied. The OOAtool CASE tool has been used to analyze the application database. A data dictionary is located in Appendix B. A complete listing of OOA results of Subject layer, Class-&-Object layer, Structure layer, Attribute layer, and Service layer are located in Appendix C.

- Advantages and Limitations of OOATool: During this thesis project, a small project MS-Window version of the OOATool has been used to perform OOA. A full capability of OOATool for other platforms, such as UNIX, MacIntosh, are also available.
  - Advantages of OOATool: After using OOATool for this application, the author suggests the following advantages of using OOATool:
    1. fully supports OOA methodology that Coad and Yourdon(34) proposed,
    2. ease of use,
    3. GUI.
  - Limitations of OOATool: From the viewpoint of the user, no tool is perfect. As for OOATool, the following limitations is observed:
    1. MS-Window small project version is limited to model only 15 classes and objects. Other versions are not limited in this point.
    2. Message connections, instance connections, and subject boundaries are randomly drawn. This makes the drawing hard to read.
    3. Once the results have been achieved by OOATool, a redraw on other drawing tool is usually required for improving the readability of OOATool drawing results.

4.5 Applying OO to develop the Application

- Apply OOD to develop the Application: The design methodology in this section is a combination of Booch(23), Rumbaugh et al(144) and Coad and Yourdon(34), (35). As described in previous chapter, the high level and low level design are incremental processes. In high level design, the steps proposed in Section 3.5 have been applied. OOD results for this three steps are located in Appendix D.
- Apply OOP to Implement the Application Database: The ITASCA C/C++ Application Program Interface (extended OOPL) allows access to objects in the ITASCA database by modeling them using ordinary C++ classes. A prototype source code using the C/C++ Application Program Interface (API) is located in the hawkeye server under *hwu/api* sub-directory.

4.6 Some Data Security Considerations

Data security incorporates three interdependent yet independent parts: secrecy, integrity and availability. Secrecy, or confidentiality, is the prevention of unwanted access or disclosure of information. Integrity is depicted as the ability to prevent unwanted modification of information. And availability refers to a system that does not prevent access to authorized users.

A secure DDBMS must include at least five components(121):

- Authorization: maintains information concerning access rights to database objects at each site. This can achieved by at least two ways: Operating support and DBMS support.
- Authentication: guarantees that database users at each site are accurately identified and authenticated;
- Access control: whether the control is distributed or centralized, the database administrators (DBA) or the system security officers (SSO) must enforce authorization rules to each user at each site;
- Auditing: keep logs of access and commands from each user at each site, especially, the security-relevant database events; (such as update data)

– Assurance: ensures that a database security system operates as secure as claimed.

Data security control is an important function of a DBMS. The techniques to implement data security include two major aspects: data protection and authorization control. Data protection prevents unauthorized users from understanding the physical contents of data. The main data protection approach is data encryption. The techniques of data protection have been widely applied, ranging from national defense to commercial secret-information protection. The second aspect, authorization control, which is more specific to DBMS is the focus of this section. In DDBMS, many users from different sites may have accessed to the database system. How to manage the authorization control when users and data have been distributed?

The DBA or SSO is responsible of taking control of data and programs accessing that data. One important function of control is granting of authorization for data access. In centralized DBMS, security problems can be caused by unauthorized reading, modification or destruction of data. It is the responsibility of the DBA to authorize system users to access only a limited portion of the database. In relational DBMS, the DBA may use some non-procedural SQL to grant or revoke the operations on the database to certain users. The operations include: *SELECT, GRANT, PROJECT, UPDATE, INSERT, DELETE*, etc. The DBA should make sure different levels of access authorization have been properly assigned to different groups of users.

Handling remote users authorization and data still remains a big issue. Remote users authorization and authentication can be achieved by the support of distributed operating system and existing DBMS platforms. In these systems, users names and passwords are replicated at all sites in the database. In these cases, local programs when initiated at a remote site must indicate the users name and passwords. However, managing different groups of users in distributed environments is difficult. These user groups can be located at other sites and data may be stored at various sites. Some existing platforms provide two-phase commit protocol and concurrency control. Non-procedural query languages help the DBA to secure the distributed system.

- The ITASCA Security Approach: The security procedures and techniques provided in the ITASCA is discussed in this section. The authorization scope in the ITASCA is composed of three domains: role, object, and authorization type. A role depicts a group of users or a single user. An object can be defined the database itself, a class, or a class instance(65). Authorization type refers to read, write, or create objects. It is the responsibility of DBA to define the role, objects, and authorization type to different users. The ITASCA system uses a rich set of authorization utilities. This includes: implicit authorization, positive authorization, negative authorization, root authorization, class authorization, instance authorization, attribute authorization, and method authorization(65). For example, granting a user access to 99 of 100 instances requires only two authorizations. First, grant a *read-all-instances*, then *deny-access-instance* for that particular instance.

4.7 Application Design Evaluation using ITASCA

For designing database application, two strategic design decisions can be made: 1) choose to use an off-the-shelf DBMS platform, or 2) design a programming database from scratch. The second alternative is not affordable because the time and cost of developing the software can outweigh that of procuring the existing platform. An off-the-shelf solution also has the advantage of being more flexible and portable since most of the popular DBMSs have implementations that can run on a variety of hardware platforms.

Many databases application that are not efficiently supported by conventional DBMSs, such as: documents in an office, software engineering projects or programs and their components, scientific data, multimedia management, and others. Choosing a database solution that satisfies the requirements of the application is essential. Object-Oriented technology is desired for the application because of the following reasons:

- Inheritance or multiple inheritance is desired to model the complex data schema in the application database. For instance, Generic-person can be defined as a superclass from which Student, Employee, Faculty, etc. can inherit the attributes.
- Object semantics: ODBMSs allow users to group related data as objects, to reference objects by object identifiers as well as traditional keys, to aggregate related objects as composite objects, and to generalize common objects through inheritance of properties.

Distributed environment is desired for the application database because of the following reasons:

- Remote access is desired in the application database since the users are distributed.
- Data are distributed and each local site maintains its autonomous data.

For DOODBMS, we have some existing platforms to help prototyping. These platforms are commercially available. One well-known platform is the ITASCA System. This system has been selected for prototyping the application for this investigation. The reasons to use this platform are: the ITASCA Systems support multiple inheritance, support dynamic modification, support object-oriented modeling, protect investment in existing code and database systems, reduce the costs of maintenance and modification, support version control, change notification, support multimedia data management, good performance features, and good facilities for cooperative engineering, etc.(5).

Other alternatives, such as ObjectStore, Gemstone, ONTOS, distributed Ingres, Oracle, etc. have been considered. These alternatives have not been selected because none of these DBMS supports both object-oriented and DDBMS features.

- Schema Change Operations: Schema modification capability is another reason why the ITASCA Systems were selected. These operations are essential to system development and administration. All the operations in the following list have been implemented in the ITASCA system(5), (65). These semantics of schema changes includes:

* Change an instance variable

  1. Add, drop an instance to/from a class.
  2. Change the name/domain of an instance of a class.
  3. Manipulate the shared value of an instance, i.e., add, change, and drop a shared value.

* Changes to a method

  1. Add/drop a method to/from a class.
  2. Change the name/code of a method in a class.
  3. Change the inheritance of a method.

* Changes to an edge.

  1. Make a class S a superclass of a class C.
  2. Remove a class S from the superclass list of a class C.
  3. Change the order of superclasses of a class C.

* Changes to a node

  1. Add a new class
  2. Drop an existing class.
  3. Change the name of a class.

This set of taxonomy of schema changes is defined in (13). Most of the OODBMS do supply part of the functions. ORION/ITASCA has the most sophisticated modifications of dynamic schema evolution facilities of any OODBMS.

- Using the ITASCA Associated Tools:

  * Dynamic Schema Editor: The ITASCA Dynamic Schema Editor (DSE) allows application programmers to directly create and modify the schema with no need to generate and load a Data definition Language. This graphical tool fully support the schema operations that are mentioned above. A C++ header file for the application is generated by the schema editor. This file must be included in the application program to achieve data persistence in the application. In this step of using the tool, the results from

OOA are mapped directly to database schemas created using the ITASCA DSE. A C++ header file for the application is generated by the schema editor to manage the database persistence and distributed functions. For more detailed information about the ITASCA DSE, refer to (68).

* Using DBA tools of the ITASCA: This tool simplifies routine database administration tasks by a graphical user interface. This tool supports appliication programmers or DBA by setting up security authorizations (such as *Enable authorization, Disable authorization, Create role, delete role, Create user, Set role etc.*), site management utilities (such as *Initialize, Restart, Shutdown, Destroy, Backup, etc.*) , database management utilities (such as *Relocate, Compress database*), examining database communications (*Disk Usage, Connections etc.*), examining and modifying object locks, and *Help*. For more detailed information about the ITASCA DBA, refer to (69).

* Using the ITASCA C++ API: This C++ Application Programming Interface (API) allows application programmers model classes and objects directly from C++ class and objects. However, this leads to as problem: data persistence. To make best use of the C++ API, use the DSE to generate a C++ header file for the persistent classes(66). This interface provides an extended OOPL environment for C and C++ programmers to develop applications that utilize the ITASCA platform. The ITASCA C++ API provides the following functionalities to facilitate application design:

  1. The persistence of objects is provided.
  2. Query utilities are provided: select, update, etc..
  3. Objects and classes can be easily implemented after objects are created using the ITASCA Dynamic Schema Editor.
  4. Various database utilities are provided: utilities such as: System administration utilities, Query Capabilities, Data management Facilities, Versioning objects, Using composite objects, database initiate, restart, commit, abort, quit, etc., refer to (66), (67) for details.

- Summary of Application Design Using the ITASCA : The ITASCA system can support this application as the following evaluations show:

    * A distributed system is favored since users of the application are distributed. Users may access the application databases through a network environment.

    * Object-oriented technology is preferred for incremental design and quick prototyping for most of the engineering applications, including the application. Because the design process is interactive: important feedback is provided to the developers or users.

    * Object-oriented modeling is suitable for modeling the schema in the application databases. Because specialization, generalization and inheritance can be applied to prototype the application.

    * Multiple inheritance is desired for this application. For instance, in the application database, a member of faculty is also a member of the Special interested group and computer support personnel etc.

    * Dynamic modification of schema is favored because it reduces the development cost of the application databases.

    * Private query and shared query for private database and shared database is desirable in the application databases.

    * ITASCA does not depend upon any single site, i.e., no single site acts as a master site, thus this system has neither a central data server nor a central name server(65). This feature is important for maintaining a robust database system with high availability in network environments. Single node of failure does not bring the whole system down. This is a desirable feature of most of the DDBMS, including the application databases.

    * ITASCA supports dynamic schema modification. This provides flexibility for application development, maintenance, and modification of the application databases(5), (65).

    * ITASCA supports version control. Various version controls include: transient version, working version, released version, and generic version(5), (65).

    - The ITASCA System is a highly advanced programming and database environment with sophisticated features, such as: active schema editor, C and C++ interface and LISP interface. This environment is designed with software reusability in mind, thus being very helpful to the application when we try to reuse the design to incorporate with a large software application.
    - Good database administration utilities: The ITASCA system provides efficient utilities for database administration. These utilities include: initialization, sites management, database management, object migration, tuning of database performance, and disk compression(65). These are desirable features in the application databases.
    - Good authorization and security management: The ITASCA system uses a rich set of authorization utilities. This includes: implicit authorization, positive authorization, negative authorization, root authorization, class authorization, instance authorization, attribute authorization, and method authorization(65). Good authorization and security control are desired in the application because information in the application databases is confidential.
    - Good user interface utilities, such as dynamic schema editor, C and C++ interface, LISP interface.
    - Good automated tools support: ITASCA DSE and DBA Tool provide efficient and convenient support for DDMS management.

- Partial Prototyping Using the ITASCA C/C++ interface: A partial prototyping of the application databases using the ITASCA System C/C++ interface to define the schema for the application databases is located in hawkeye file server under *hwu/api* sub-directory.
    - Comments after Prototyping:
        1. This partial prototyping is to design the schema using the ITASCA C/C++ interface. Among the many OOPLs, C++ has been selected because of the powerful capabilities.

2. The benefits of using object-oriented modeling to define the schema for the application database are obvious, i.e., multiple inheritance is used to model the schema through superclasses and domain declaration.
3. Persistent methods are better defined using the DSE. This tool generate header files for object persistent. However, using this tool to create persistent methods requires the knowledge of Lisp programming techniques. This could be a learning curve to non-Lisp programmers.
4. Suppose that the application has been designed and implemented on a centralized, relational DBMS platform or programming languages, then, there comes a decision that the senior managers and technical personnel have to make: either continue using the existing system or adopt a new technology. The major consideration will be based upon the policy, benefits and costs. Suppose that the new technology is favorable, we need to migrate the data on the old system to new environment by some mechanisms. The start-up costs may be high for starting from beginning to figure out how to interface object-oriented application with non-object-oriented ones(23). However, once the platforms and techniques have become familiar to the developers and the users, the benefits will show up very soon.

4.8 Summary

The purpose of this chapter is to apply the methodology presented in chapter 3 for designing the thesis application. Although the methods are organized into steps, it does not mean that these principles are to be applied in a rigid sequence. Despite the fact that some steps in the design process must precede others, the nature of object-oriented paradigm is iterative and incremental. In this iterative and incremental process, the author started by 1) performing OOA to define the boundary of the problem domain, 2) evaluating design considerations, and 3) evaluating selected platform and CASE Tools to achieve a rapid-prototype design.

V. Detailed Design and Implementation

5.1 Introduction

The objective of this chapter is to discuss the detailed design and implementation of the database application using the ITASCA ODBMS, the ITASCA Dynamic Schema Editor, the ITASCA DBA Tool, the ITASCA C++ API, and the OSF/Motif X Windows Toolkits.

5.2 Using the ITASCA System to Prototype the Application

With the aids of the ITASCA DODBMS platform, much of the functions have been provided. Some major functions includes:

1. Provide an object-oriented data modeling power.
2. Persistence of data storage.
3. Distribution functions.
4. Query language: high-level access to attributes.
5. Concurrency control.
6. Transaction management.
7. Recovery: media failure, site failure, communication failure, and crash recovery etc..
8. Security: access authorization.

These functions help prototype application quickly by giving the developers more powerful capabilities to manage objects. Detailed implementation of the thesis application's connections to the ITASCA platform is located in Appendix F.

- The ITASCA Dynamic Schema Editor (DSE): After using the schema editor, the following advantages associated with using this tool is observed:
    - capable of modifying schema without shutdown database,
    - fully support the functions of the above taxonomy of schema change,

- capable of opening or closing session, login with different user name, commit, or abort transaction.
- Good private database management: Create, compress, get owner, move private database to global database,
- schema browser supports the functions of the taxonomy of schema changes listed above.
- ease to use,
- GUI: this tool is designed using OSF/Motif X Windows Toolkits,
- generate C++ header file for C++ application programs.

Along with advantages, the following limitations associated with using this tool is suggested:

- some command take an "instance" to execute, such as authorization of access of some classes to a super user role (Create user in *super user role* is suggested to avoid potential conflicts(66)).
- untraceable error may happen owing to the system is waiting for a lock in distributed transaction.
- segmentation fault and bus error happen frequently when using the ITASCA Schema Browser to make new classes and define attributes, methods or add superclass to a new object.
- The problem of cyclic class definition is hard and tedious to avoid when generating header file for the application.

- The ITASCA DBA Tool: After using the DBA Tool, the following advantages associated with using this tool is observed:
  - capable to support DBA by performing sufficient database maintenance and administration jobs, such as: site management, security management, and data management.
  - capable of opening or closing session, login with different user name, commit, or abort transaction.

- good private database management facilities: create, compress, get owner, move private database to global database,
- ease of use.
- GUI: this tool is designed using OSF/Motif X Windows Toolkits,

Along with the advantages, the following limitations associated with using this tool is observed:

- Some commands take an "instance" to execute, such as create super user role (Create user in *super user role* is suggested to avoid potential conflicts(66)).
- untraceable error may happen owing to the system is waiting for a lock in distributed transaction.
- authentication management (such as: create or delete role, enable or disable authentication) is very slow.

– Summary of Using the ITASCA System: The ITASCA System is an advanced commercial DOODBMS. This system has features belonging to common database systems. These features include: concurrency control and locking, transaction management, composite objects, dynamic schema evolution, versions, multiple security levels, and persistent storage for data and schema. In addition, ITASCA has many useful functions that deal with distributed environment. These functions are: shared and private databases, distributed version control, distributed transaction management, distributed query management, distributed change notification, long-duration transactions, dynamic schema modifications, multimedia data management, etc..

After using the ITASCA Systems to prototype the thesis application, the following advantages and disadvantages of using the ITASCA were suggested:

- Some Advantages of Using the ITASCA: After using the ITASCA system during the thesis efforts, the following advantages using ITASCA system were observed:

  1. System Capabilities: object-oriented modeling, distribution function.

2. Good Utilities: Extended OOPL, query utilities, and persistent object storage.
3. Good Administrative Facilities: graphical DBA tool using OSF/Motif user interface.
4. Dynamic system configuration: graphical dynamic schema evolution facilities.
5. Reduce development time and risk: the ITASCA system is suitable for rapid-prototype.

* Some Disadvantages of Using the ITASCA: After using the ITASCA system, the following disadvantages were observed:

1. System-dependent Knowledge: In OODBMS, various approaches ware adopted among different systems. For example, query facilities various among different OODBMS systems. The approaches in programming with API in ITASCA, for example, is system-dependent. No other system adopts the same approach.
2. Complicated application programming interface.
3. When executing, this platform takes a huge size of the memory.
4. Documentation is limited to front-end users level, not enough information for application developers.
5. Execution is slow.
6. When using C++ API, it is difficult to incorporate with OSF/Motif because of the Motif programs are not yet designed in C++.

5.3 An X Window User Interface Design

The purpose of the user interface is to facilitate user-computer communication by enveloping hardware and software, particularly the semantics of application, in a dialog(1). This dialog hides the structure of textual input and output devices, operating systems, network, and applications, and let user switch application rapidly, unencumbered by technical mechanisms.

Application design with X Window System has several advantages(151), (1), (51), (110):

- It results in better application user interface and user control.
- Design can be rapidly prototyped and implemented, possibly user interface can be implemented before the application code is written.
- It is easier to incorporate changes discovered through user testing.
- Powerful capabilities, such as file selection panel, can be easily included.
- Different applications will have more consistent interface.
- The interface code will be easier to create and economical to maintain (reuse).

Figure 5.1 (135), (1) shows the organization of user-interface software in the implementation-oriented model. The application program has access to the operating system, the window management system (some consider as an extension of the operating system) and graphic package, the toolkit, and the user interface management system. The interactive design tools allow non-programmers to design such widgets as menus and dialog boxes, as well as dialog interaction sequences.

As information systems grow and increase in complexity, the problems faced by the users in understanding the data content and in trying to retrieve relevant data quickly and easily become acute. If end-users are going to be able to exploit their information resources, then it is essential to provide user-friendly query methods. Therefore, user-centered design becomes important. User-centered design focuses not on technology, but on the user's cognitive abilities and professional or personal preferences in the applications. In database application design, there is a growing interest in developing query languages based on iconic graphical user interfaces using standards such as OSF/Motif, Open Look, and X windows.

With the arrival of multi-window systems built on bit-mapped display, the user expects much more from a user interface than they did when only CRTs were available. However, a good user interface is generally time-consuming to build because it has to deal with low-level tasks, such as I/O. Once the user interface was

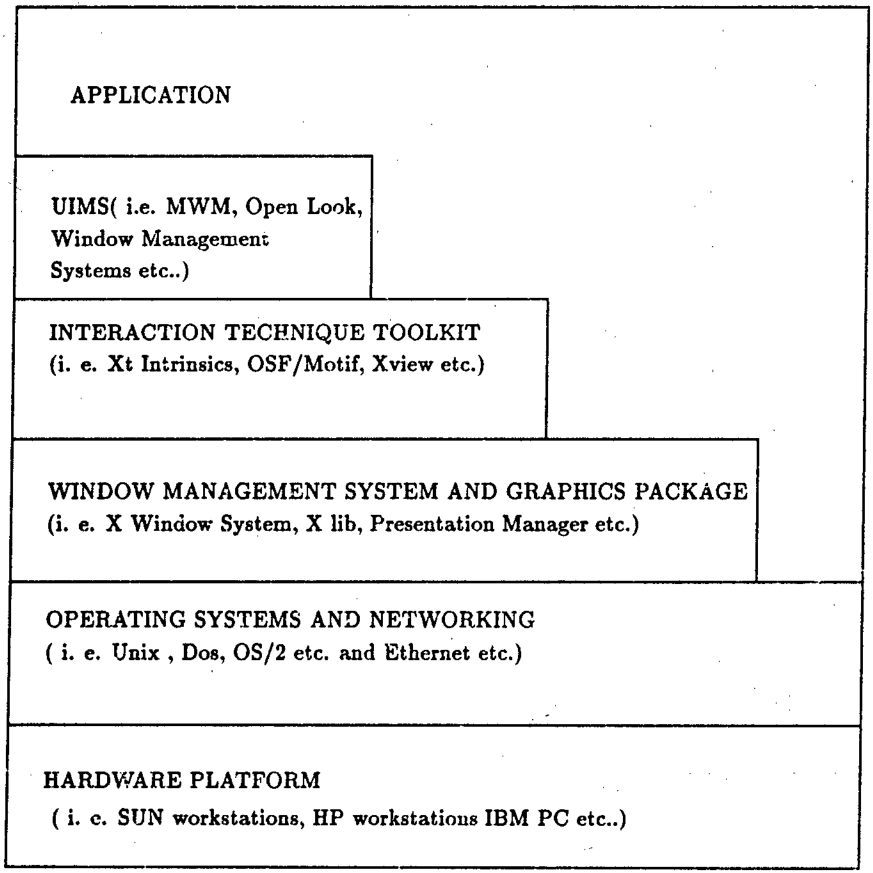

APPLICATION
UIMS(i.e. MWM, Open Look, Window Management Systems etc..)
INTERACTION TECHNIQUE TOOLKIT
(i. e. Xt Intrinsics, OSF/Motif, Xview etc.)
WINDOW MANAGEMENT SYSTEM AND GRAPHICS PACKAGE
(i. e. X Window System, X lib, Presentation Manager etc.)
OPERATING SYSTEMS AND NETWORKING
(i. e. Unix , Dos, OS/2 etc. and Ethernet etc.)
HARDWARE PLATFORM
(i. c. SUN workstations, HP workstations IBM PC etc..)

Figure 5.1. The User Interface Implementation-orientd Model

decided by the users, the predefined widgets (windows objects) are instantiated and organized to build the user interface.

Building user interfaces using predefined widgets in the X11 with OSF/Motif toolkits are difficult and time-consuming. Programming with widgets consisted mainly of declaring several types of widgets (such as command widgets, label widgets), attaching them to an appropriate parent widget, setting the size and position of the windows, and linking the routine that should be invoked when a particular widget is selected. This toolkit provides mouse-pointing facilities, text widgets, and several other features that make implementing a user interface easier. Composite objects, such as pull-down menus, are implemented by declaring several command widgets and attaching them to the parent menu widget. The design of user interface is ideally independent to application design. Technical considerations and user preference are two distinct stages of application design. The fact that user acceptance of the application plays an important role to the success (acceptance) of the application. The application developer must not neglect the user interface design.

On the graphical user interface, each button pressed represented a routine call (notify sub-program to execute). So during the application design phase, with regard to user interface concern, the "called by" should be specified by the designers to allow user interface to better communicate with the computational application.

Figure 5.2 shows the framework for user interface design. The objective of the user interface design using X Window Systems is to improve the modularity of a program by designing each action as a program unit (procedure, sub-program). Each action can be a widget or a menu-bar in the window frame.

To implement the prototype user interface, the OSF/Motif toolkit was selected based on the following characteristics:

1. Capabilities: this graphic interface was capable of presenting sufficient graphical representations for the application users.

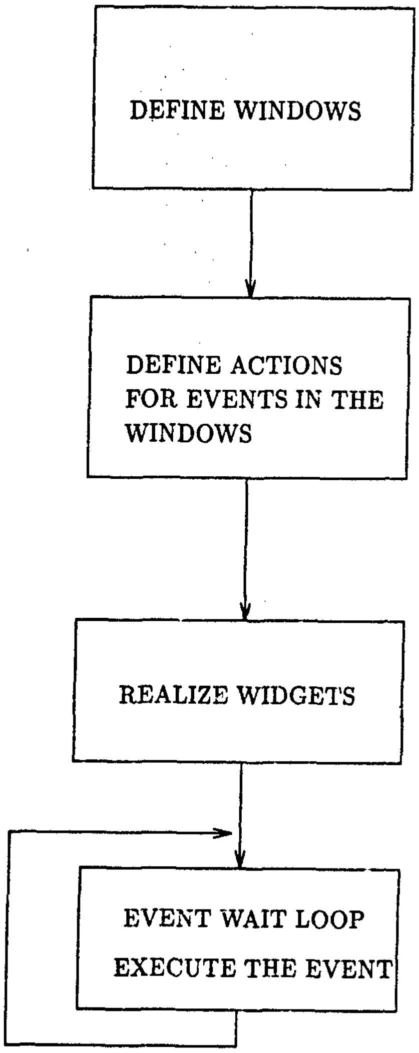


Figure 5.2. A Framework for X Window System User Interface Design

2. Portability: this "industrial standard" interface are able to be demonstrated on as many platforms as possible.

3. User friendliness.

4. Simplicity for use: this graphical user interface was easy to use.

5. Availability: this graphical user interface is widely used.

Some presentation managers such as: Open Look (Xview), have been under considerations. Assuming that the issues of capabilities, user friendliness, availability, and portability are the biggest concern of the system, it explains why OSF/Motif was selected.

- Depiction of Function Calls: Depiction of the function calls (relating to user interface functions): In this process, three functions were specified: 1) sequential functions, 2) hierarchical functions, 3) functions calling other functions. Function calls from the user interface are achieved by a sequential case statements. Function calls are the call-back functions, which are triggered from the user interface when users select the menu bar items from the pull-down menu. Each case statement is represented by an integer number defined in the header files. A detail description of application function calls relating to user interface functions is located in Appendix G.

- Using OSF/Motif: Although various toolkits exist, OSF/Motif is one widget set commonly chosen by application programmers as a part of their X Toolkits. X Toolkits is comprised of a "layer known as the Xt Intrinsics, and a set of user interface components known as widgets"(181). A widget set is a predefined (such as by the user or some other source) set of objects that can be displayed on a screen, such as buttons and menus. A standard widget set may restrict the user look of the application, but this is still consistent with the goal of X Window System by providing mechanism, not policy(74). The application program make functions calls directly to the widget set, the Motif widget set may make calls directly to the Xt Intrinsics.

Figure 5.3 shows the first cut of the top-level menu of the user interface (images) for the thesis application. Figure 5.4 illustrates the widget hierarchy of the Motif user interface application. The highest level tasks of the application user interface are categor:.ed as: *D.B Utilities, Retrieve, Create, Query, Update, Print, Info, File, Edit, View, Options, Others,* and *Help.* Each highest level task is represented in a push button, which is consists of a pull-down menu to invoke lower level tasks. Highest-level tasks, such as *Create, Query, Update, Print,* have a similar user access pattern. This access pattern shares a similar lower level pull-down menu. The user input values on these screen form are identified by pointers which pointed the Motif text widget fields. These values of these fields are entered by the user. A detailed implementation results are located in Appendix H.

- Advantages of Using Motif: The following lists the advantages of using OSF/Motif(74):

  1. Motif provides a standard interface with a consistent look and feel. Your users will have less work to do in learning other Motif applications, since much of the work learning other Motif applications will translate directly to your applications.
  2. Motif provides a very high-level object-oriented library. You can generate extremely complex graphical programs with a very small amount of code.
  3. Motif has been adopted by many of the major players in the computer industry. Many of your customers are probably using Motif right now. You'll do a better job selling to them if your applications are also based on Motif.

- Disadvantage of using Motif: From a different viewpoint, the following disadvantages were observed after programming using Motif during the thesis effort:

  1. Large learning curve: due to the powerful capabilities and complexity of the system. (Motif requires explicit and extensive coding to create the interface.) Learning curve does effect decision of using such a complex system. The learning curve model (77) is suitable to measure and evaluate tool adoption.

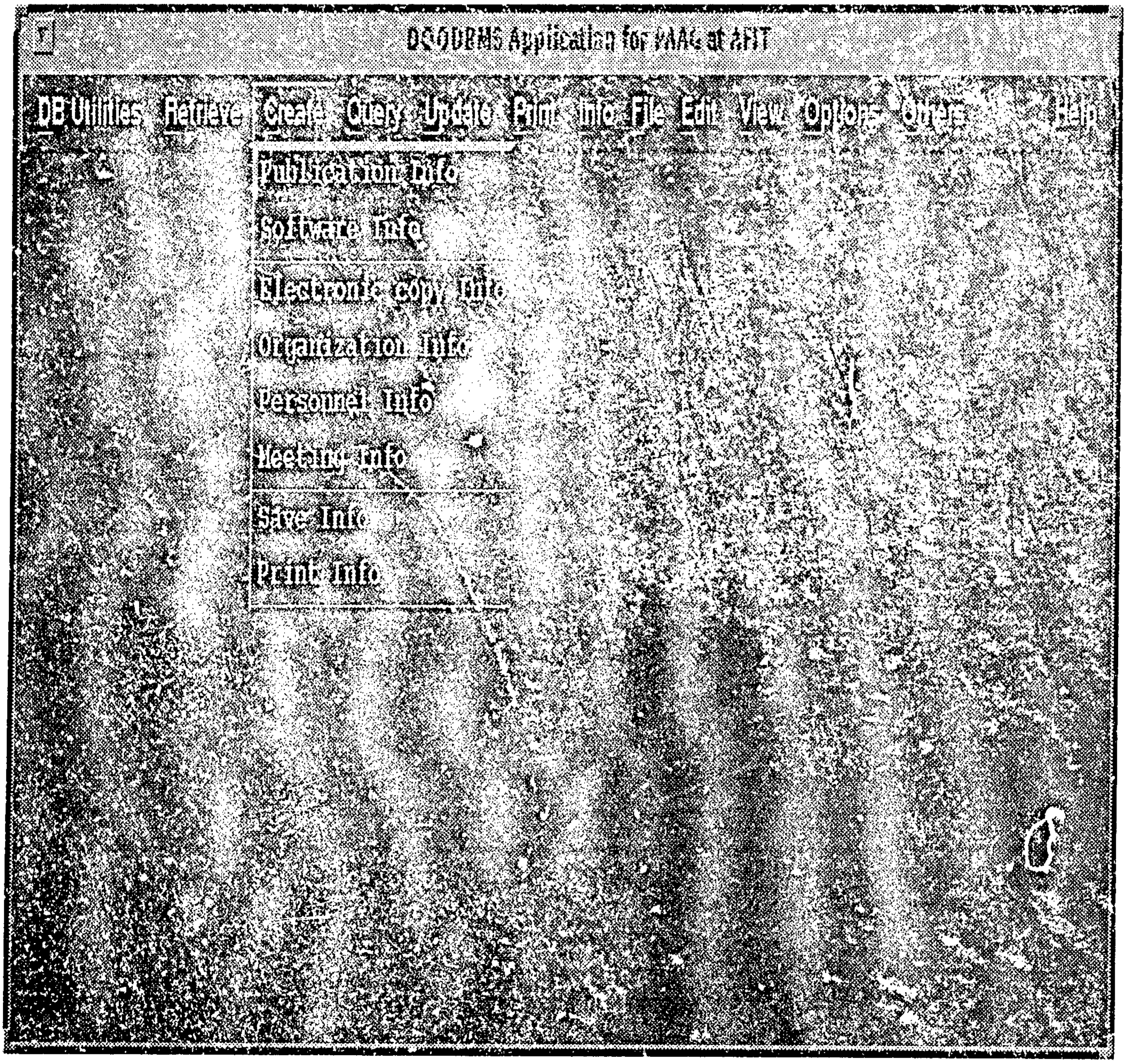


Figure 5.3. The DOODBMS Application's User Interface: Top-level Menu

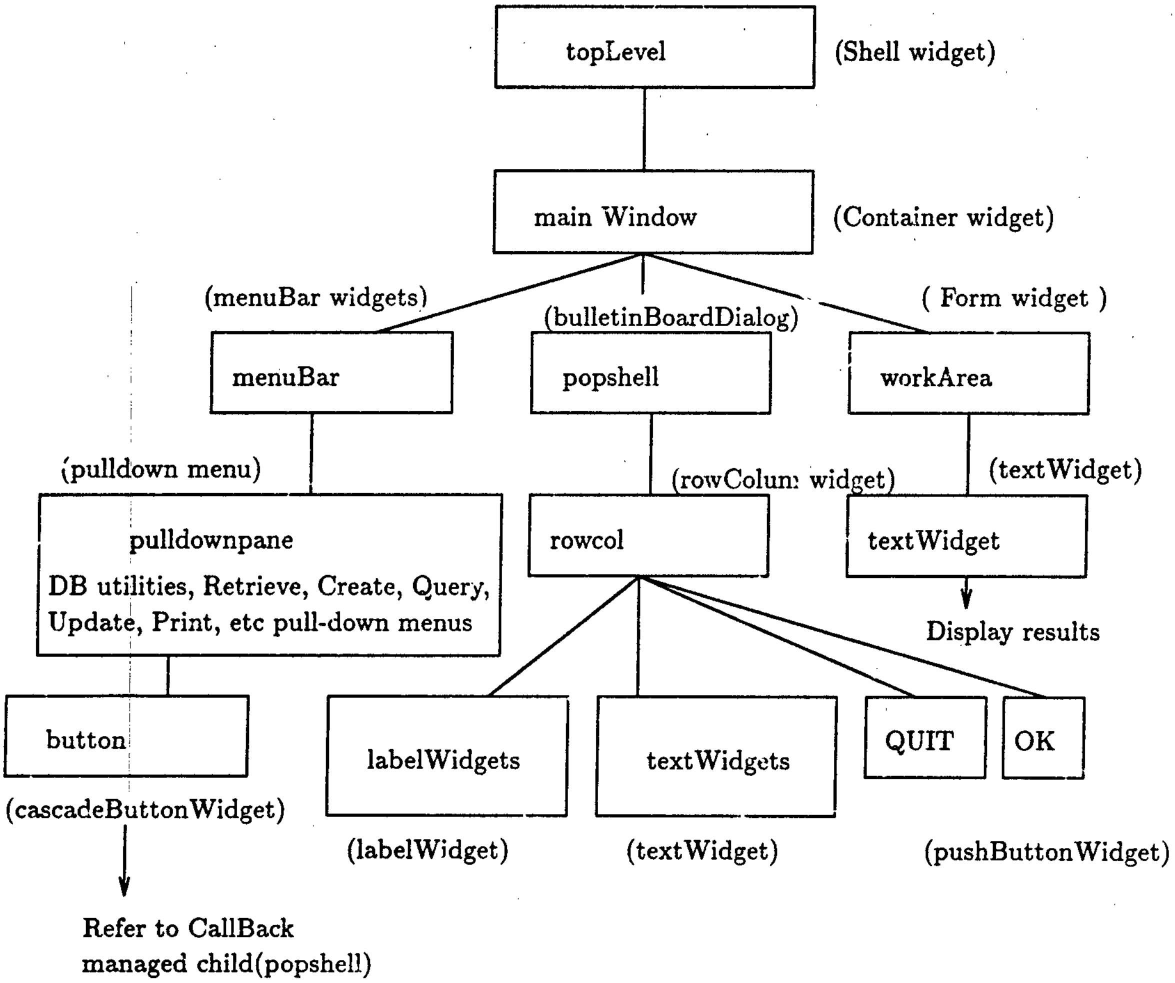


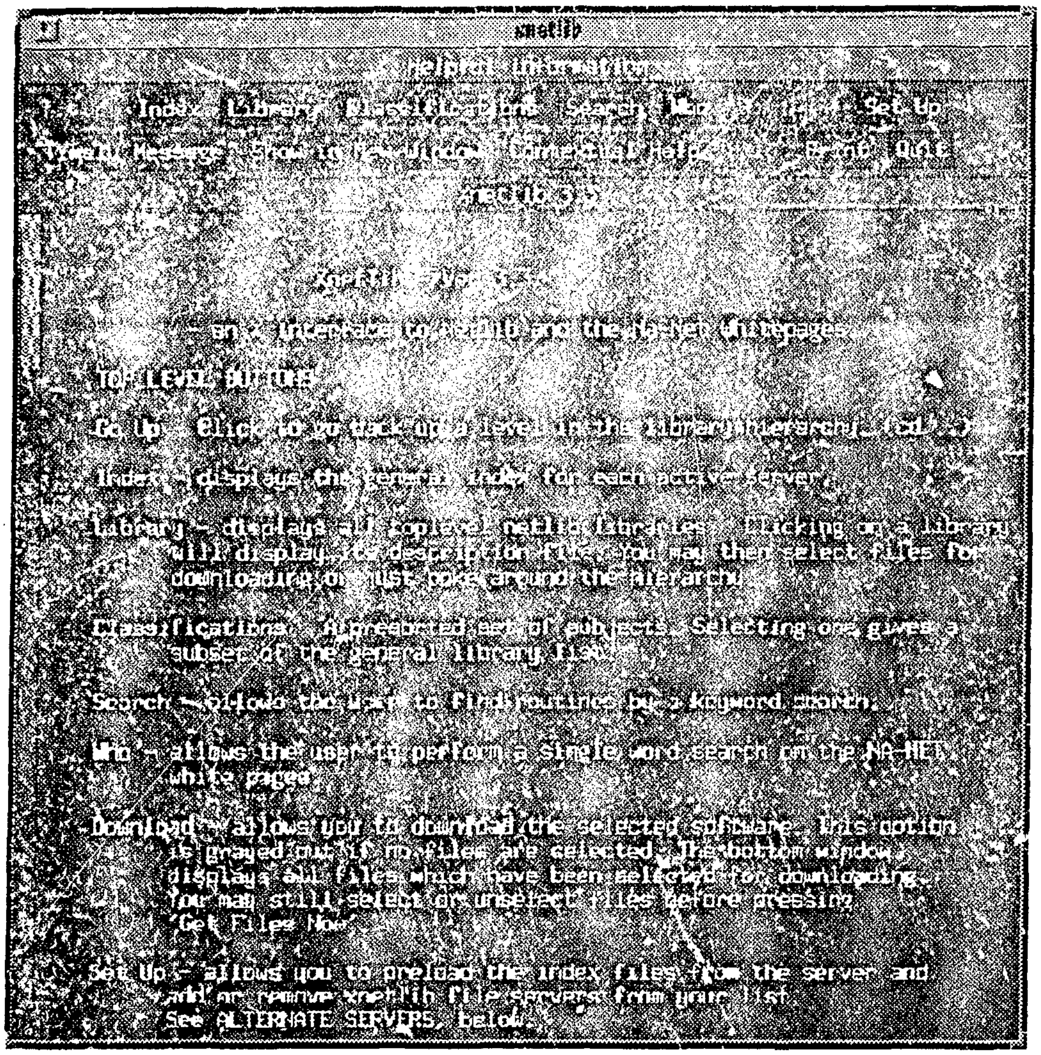
Figure 5.4. Widgets Hierarchy of the User Interface Application

2. Motif is only available for X-Windows based computers.
3. Motif programs consume huge memory space of the system and the execution speed is slow.

- Modify and Incorporate the Xnetlib: The Xnetlib is a new version of netlib developed at the University of Tennessee and Oak Ridge National Laboratory in the late 1992. Unlike netlib, which uses electronic mail to process requests for software, xnetlib uses an X Window graphical user interface and a socket-based connection between the user's machine and the xnetlib server machine to process software requests(180). This software aids the PAAG in accessing the latest softwares that are available form the Oak Ridge National Laboratory. This hierarchical window system can provide area of interests for users to retrieve data from the database. In order to incorporate this software to the application, an attempt was made to modify the program to incorporate to OSF/Motif widget. The interprocess communication is achieved by using two-way pipe. Thus, the data retrieved from using Xnetlib is displayed in the PAAG user interface text field. Figure 5.5 lists the Xnetlib user interface.

5.4 Integrating Application with User Interface

Application design is, ideally, independent of the user interface design. During the process of application design, however, specifying the procedure or subprogram calls from the user interface could be beneficial to application design. This benefit includes: quicker design, modularity, easier maintenance etc.. The user interface can be generated or coded before any application coding occurs. This is achieved by adding in the null functions in the user interface programs. Because the access pattern of database application is not complicated, a typical access pattern of database application is similar to SQL constructs, such as *SELECT*, *UPDATE* etc.. The user usually attempts to access, change, print, save, or add in information, especially in the targeted system, graphical user interface hides access and query constructs of a DBMS into graphical representations in the following ways:



xnetlib 3.[illegible]

Xnetlib - Ver [illegible]

- an X interface to netlib and the NA-Net Whitepages

TOP LEVEL BUTTONS

Go Up - Click to go back up a level in the library hierarchy. (cd ..)

Index - displays the general index for each active server.

Library - displays all toplevel netlib libraries. Clicking on a library will display its description file. You may then select files for downloading or just poke around the hierarchy.

Classifications - A presorted set of subjects. Selecting one gives a subset of the general library list.

Search - allows the user to find routines by a keyword search.

Who - allows the user to perform a single word search on the NA-NET white pages.

Download - allows you to download the selected software. This option is grayed out if no files are selected. The bottom window displays all files which have been selected for downloading. You may still select or unselect files before pressing 'Get Files Now'.

Set Up - allows you to preload the index files from the server and add or remove xnetlib file servers from your list. See ALTERNATE SERVERS, below.

Figure 5.5. The Xnetlib User Interface

- retrieve data items can be built into graphical representation with user input text field.
- data selection panels allow user effectively select data items for performing operations, such as delete, select, and print data items effectively.
- Create and update data items can be implemented as user input screen forms.
- prompt user input with appropriate graphical prompts for effectively access data items, instead of letting user conceive access constructs of the DBMS, thus eliminating the learning curve of the query langt.age constructs for the non-professional users and thus increase access efficiency.

Integrating the application programs with user interface is achieved by using multi-processes communication mechanism in user interface callback. These communications (data passing) between application and user interface can be achieved by using any of the following ways of programming techniques(157) with the Unix operating system support: 1) use two-way pipe structure to pass data, 2) use message queue to store data, and 3) use shared memory, 4) FIFOS, 5) use socket (for distributed processing). There are trade-offs among these techniques (such as pipe structure being the easiest way to pass data between processes, suffers the possibility of process starvation), the reader is referred to (157) for more detailed explanations.

- Problems Encountered: During the integration, several challenges occurred. These are:
  1. Until this point of time, few, if any, text books cover DOODBMS design and implementation. Most of the text books cover either DDBMS (assume relational data model) or OODBMS (without distribution concerns). The dependent knowledge from certain commercial platforms does not provide sufficient design documents for application developers. A better solution might be the training courses provided by the related vendors.
  2. The ITASCA DSE segmentation fault and bus error: this problem happens when using Schema Browser to make new classes and define attributes, methods, or add superclasses to a new object. The current release of DSE

has not yet solved this problem. The technical support claimed that the problem will be solved by the next major release(177).

3. The ITASCA DBA Tool has untraceable errors that happen frequently owing to the system is waiting for a lock in distributed transaction.

4. The ITASCA DSE class definitions easily incur cyclic class definitions due to the system assumed the all classes created in the ITASCA inherit the superclass called *CLASS*. When a class is created without any superclasses, it defaults to having *CLASS* as its superclass. Later when you define a second class (again without specified superclass) it also has superclass *CLASS*. Now if you use *add-superclass* to make the first a superclass of the second, you have a cycle through CLASS. The recommended solution at this time is to use *remove-superclass* to remove CLASS from the superclass list of the "child" class at the time you perform the add-superclass(177).

5. The CenterLine C++ preprocessor does not preprocess the header file that generated by the DSE tool very well. Error messages indicate that the header file is not compilable.

6. The CenterLine C++ preprocessor does not support commercial Quest Window's Motif programs very well. When Motif's programs are preprocessed by CenterLine C++, error messages indicate that the include files of Quest Windows is not compilable.

7. Most of the Motif programs were written in C, without the intention of incorporating C++ features, few, if any, C++ applications are incorporated with Motif. This research domain remains a new ground(105).

8. Dynamic binding of C++ programs are hampered by some unresolved problems of SUN workstations. One example is that the *xfig* (figure drawing package) of SUN workstations suffers run time error when linking the library dynamically(147). Another example is that some Motif programs are not working when linking the library dynamically. One solution is to link the library statically.

9. Compilation error messages of C/C++ programs, such as segmentation faults, bus error, unresolved inheritance problems, etc., are hard to trace. Neither these messages are well-documented in the related manuals. One solution is to use the debuggers of the SUN workstations, such as *dbxtool, xdbx.*

5.5 Summary

Designing and implementing a DOODBMS application is difficult and time-consuming. Recent advances in DBMS platform, OOA, OOD, persistent object-oriented programming languages, and other associated CASE tools supporting database application development have reduced the burden of developing DODBMS application. In the targeted application, the ITASCA system provides various enhanced functionalities, such as fully distributed capabilities and object-oriented modeling, and tools that together does reduce the development time of the system. Meanwhile, The OSF/Motif user interface design and implementation is also difficult and time-consuming. The trade-offs are the user-friendliness, capabilities and portability that it can provide.

VI. Conclusions

6.1 Introduction

The purpose of this chapter is to summarize the thesis and recommend actions based on the lessons learned during the thesis effort. This thesis effort encompassed three major areas: 1) DDBMS design, 2) OOA, OOD, and OOP, 3) OSF/Motif user interface design and implementation.

6.2 DDBMS

With increased progress in networking technology, greater economic pressure for company and even inter-company cooperation, DOODBMS has become one current trend in the database systems development. Because a DDBMS provides more flexible and efficient data processing, research in OODBMS has also revealed an abundance of additional benefits that cannot be provided by traditional DBMS (such as data location transparency).

A DOODBMS is preferred when: 1) organizations tend to spread among a large area, 2) the users in each site have the need to insure the information is reliable and available at low cost, 3) the users need to achieve a higher local autonomy, and 4) the users want to interconnect existing database systems with a higher performance, and 5) complex data model is desired to efficiently manage data.

During this thesis effort, the following were achieved: surveys of DDBMS and OODBMS with key features comparisons, evaluations of the ITASCA DODBMS and associated tools, and proposition of a guideline for DOODBMS application design.

6.3 Object-oriented Paradigm

The problem of "software crisis" led to a rethink in the software process. Object-orientation is a good solution for the problem. Object-oriented paradigm is composed of object-oriented analysis, design, and OOP methods. Although, the boundary of analysis and design in Object-oriented is inconsistently defined. Nevertheless, the "what" and "how" distinction is important. This distinction should raise the awareness level of the OO developers and it should help remind them that requirement modification works during development are to be checked with the customer/client.

This technique of modeling provides the conceptual framework for OOD. Although there are a great diversity of methods that are advocated in this area, the principles concepts are almost the same only the notations, elaborateness, and integration within the life cycle is different. It is advantageous because each user can choose what their preference. A developer should be able to be as formal as necessary for a particular task. Thus a method should not put a limit on achievable formality. Consequently its basic concepts should be clear, unambiguous and have preferably a formal semantics.

In OOP, the major principles include abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence. Principles such as inheritance, abstraction, and encapsulation enhance the development process by allowing extendibility, reducing complexity, and increasing reusability. Object-oriented programming languages embody these ideas that narrow down the gap between object-oriented programming and database application design. The object-oriented model of the database management system is more flexible, extensible, and able to solve more difficult applications than traditional models.

During the thesis efforts, the following were achieved: survey and comparisons of various OO modeling techniques and OOPLs, evaluation of Coad and Yourdon OOA and OOD methodology and associated tool.

6.4 User Interface Design

The graphical user interface has been accepted as one of the most important parts of user-centered, interactive software development. The user interface design is not gold-plating. It helps the user focus on the important aspects of the application rather than on the user interface itself. The user interface designer provides a bridge for the user and the application. Distributed applications run a distributed set of clients. A GUI gives the user the capability to interact with a number of application clients at the same time. X Window System with OSF/ Motif toolkits is a popular window system for dealing communication protocol. It provides network transparency via a graphical user interface.

The X Window System is a software environment designed for engineering workstations. The system offers a rich and complex environment to the users. The central dogma of the X Window System is that the base window system provides mechanism, not policy. These characteristics make X Window System becomes most of the application developers choice. The graphical interface makes a computer more accessible for non-technical users who find command-driven interfaces too cumbersome. The graphical interface was more intuitive and prompted the user to make a choice, rather than remembering what they should search for. The X Windows concepts are important for our design considerations because it can concurrently support many applications based on a variety of operating platforms. Data can be updated in one view while being viewed from different perspectives in other windows. It represents an ideal solution for the needs of multiple users accessing several applications distributed around the enterprise. X also provides an excellent base for object-oriented programming.

During the thesis efforts, the following were achieved: Xnetlib was modified and incorporated for supporting the PAAG's research, X Window System with OSF/Motif user interface was evaluated and implemented, database query language constructs were hidden by the graphical user interface.

6.5 Lessons Learned

Throughout this thesis effort, the following valuable lessons were learned:

- DDBMS incorporating object-oriented technology with the X window System does enhance interoperability.

- Object-oriented technology does relieve software crisis by enhancing interoperability, solving complex systems, providing easily modified and extended solutions, and enhancing reusability.
- Although a great diversity of methods are advocated in OO technology, the principle concepts are the same; only the notations, elaborateness, and life cycle integration is different. It is advantageous because the user can choose what they prefer. A developer should be able to be as informal as necessary for a particular task. Thus a method should not put a limit on achievable formality. Consequently its basic concepts should be crystal clear, unambiguous and have a formal semantics.
- Object-oriented technology provides a consistent model of representation across all programming problems. The improvement of standardization of methodology, terminology, and technique is desired to make a uniform model mature.
- User interface is necessary for better user-machine communication in application design.
- The X Window System with OSF/Motif Toolkits allows the application developer not to be restricted in different platforms.
- The X Window System with OSF/Motif Toolkits is suitable for distributed object-oriented application design.
- User interface could be designed before any application development.
- Application design is, ideally, independent of the user interface design. During the process of application design, however, specifying the procedure or subprogram call from the user interface could be beneficial to application design. This benefit includes: quicker design, modularity, easier maintenance etc.. The user interface can be generated or coded before any application coding occurs.
- It is efficient to hide query language constructs into graphical user interface representations.
- Graphical user interface does not be necessarily free of text input.

- The pitfalls of commercial products on which all systems look very much alike: they proclaimed that their products can do everything and solve all problems(12). A key point is that users should not expect that the software product is robust especially when the product is progressing. Vendors usually release their software products to allow users to participate and react.
- Future Research in DBMS: The next generation of database applications will be larger and more complex. In particular, the amount of data will be greater, and object-oriented technology has been proposed to support extended data types, complex objects, multi-media and rules processing. Organizations will need to integrate large amounts of data in a distributed, heterogeneous environment. The problems of size, security and consistency are difficult to manage. This trend is likely to increase interest in object-oriented techniques in database management.

6.6 Recommendations

Throughout this effort, the author would like to suggest some recommendations for the follow-up students that would do research related to this area:

- Suggest using a application development Tool (CASE tool) with class library, OOA, OOD drawing.
- Use the latest ITASCA DODBMS platform: The Dynamic Schema Tool of the current ITASCA platform is hampered by class cyclic definition and segmentation fault and bus error. The next major release will correct this problem.
- Suggest using a dynamic schema on-line editor (such as the ITASCA ADE) with the ITASCA DODBMS (multi-users and multi-sites license), and the ITASCA DSE latest release of software, the ITASCA DODBMS licensed on hawkeye is currently a single user system with DSE and the DBA Tool support.
- Suggest using an X Window Builder: X Window builders come in two forms: those that can be used to design the interface only (GUI builder), and those that go beyond this to give support for application code (User Interface Management

Systems). There are several commercial and non-commercial tools of both kinds that will support the Motif interface. Some free software are suggested in the Motif-FAQ from MIT (ftp export.lcs.mit.edu):

* UIMS: WINTERP[1] , Serpent[2], The Widget Creation Library[3]
* Some commercial products are UIM/X GUI Builder[4], X-Designer[5], and ezX User Interface Management System[6].

6.7 Conclusion

Application development has always been the most significant cost for the development of new applications. From the task of feasibility study, user requirement analysis, system requirement analysis, software requirement analysis, design requirement analysis, data modeling, design, implementation, coding, testing, integrating, modification, any effort to mak : the task more efficient is desired and welcomed.

Object-Oriented paradigm is a proven success in helping the software engineers relieve the problems of software crisis. The success of object-oriented paradigm in the application environment can be enjoyed in the DBMS as the two areas continue to evolve in co-dependence. An OODBMS can provide system's extendibility, software reusability, and representation of complex data structures. Organizations (especially in academic environments) must consistently look at new technology to find better, faster, and more efficient ways to process information within that organization. Especially, the need to integrate large amounts of data in a distributed, heterogeneous environment. The problems of size, security and consistency are diffi-

[1] This Builder is free for the public. The current source, documentation, and examples can be obtained via anonymous ftp from host export.lcs.mit.edu: in directory contrib/winterp.

[2] The software is free for the public. The current source, documentation, and examples can be obtained anonymous ftp) from ftp.sei.cmu.edu.

[3] The distribution is available in several ways. The preferred approach it for you to get the compressed tar file using anonymous ftp from: export.lcs.mit.edu. The software is located in /contrib/Wcl.1.06.tar.Z.

[4] Visual Edge Software Limited, 3870 Cote Vertu, St Laurent, Quebec, H4R 1V4, Phone: (514) 332-6430, Fax: (514) 332-5914

[5] From Imperial Software Technology in the UK. Email address is sales@ist.co.uk. (+44) 743 587055

[6] Sunrise Software, International 170 Enterprise Center Middletown, RI 02840 401-847-7868

cult to manage. This trend is likely to increase interest in object-oriented techniques in database management.

Also, most people realize that if an application has a user interface that is "unfriendly" to use, it is probably going to sit on the shelf unused, therefore, user interfaces using some types of windowing system has become a common trend of most computer systems. As a result, users (either application professional or novice) tend to expect application programs to have a user-friendly, yet efficient user interface.

A DOODBMS environment has been demonstrated by many researches as the most efficient and practical way to process information in the future. Nonetheless, Object-oriented software development technology will impact the market in the 1990s much the same way as structure analysis design did in the 1980s. DOODBMS is still in young discipline. How to partition files to achieve an efficient usage of data? How to allocate file fragmentation to minimize costs of storage, transfer? These questions are yet to be fully examined.

A DOODBMS application applies object-oriented technology with distribution functions by allowing users access the DBMS through a graphical user interface is a very popular trend in DBMS application design. The computer industry has been rapidly progressing ever since the first computer was built in the 1950's. The software practitioners are finding the challenges of standard for standards, model for models, different hardware platforms, different software platforms, different operating platforms, different methodologies, different programming languages, and different policies. Using object-oriented and X Window technology for distributed environments could be an answer.

Appendix A. Terminology

This appendix defines some commonly used terms in distributed object-oriented database management system (DOODBMS). The glossary in this area are divided into three parts: 1) Terminology of Object-oriented Paradigm, 2) Terminology of DOODBMS, and 3) Terminology for Network Communications.

1. Terminology of Object-oriented Paradigm:

   - Attribute, instance variable: any property, quality, or characteristic that can be ascribed to a person or thing(34). An attribute is some data (state information) for which each Object in a Class has its own value(34).
   - Object: an abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about or interact with it; an encapsulation of Attribute values and their exclusive Services(34). Something you can do things to. An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class. The term instance and object are interchangeable(23).
   - Subject: a Subject is a mechanism for guiding a reader (analyst, problem domain expert, manager, client) through a large, complex model. Subjects are also helpful for organizing work packages on larger projects, based on initial OOA investigations(34).
   - Class: a description of one or more Objects with a uniform set of Attributes and Services, including a description of how to create new Objects in the Class(34). A set of objects that share a common structure and a common behavior. The term class and type are usually (but not always) interchangeable; a class is a slightly different concept than a type, in that it emphasizes the importance of hierarchies of classes(23). A class is specified as a means of grouping all the objects which share the same set of attributes and methods(84).
   - Class-&-Object: a term meaning "a Class and the Objects in that Class"

- Single Inheritance: a class inherits attributes and methods from only one class.

- Multiple Inheritance: a class inherits attributes and methods from more than one class. In this case, the classes form a rooted directed graph, sometimes called a class lattice.

- State: one of the possible conditions in which an object may exist, characterized by definite quantities that are distinct form other quantities; at any given point of time, the state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties(23).

- Identity: the nature of an object that distinguishes it from all other objects(23).

- Behavior: how an object acts and reacts, in terms of its state changes and message passing(23).

- Method, Service, message, or Operation: an operation upon an object, defined as part of the declaration of a class; all method and operations are usually interchangeable(23).

- Subclass: a class that inherits one or more classes (which are called its immediate superclass)(23).

- Superclass: the class from which another class inherits (which is called its immediate subclass)(23).

- Structure: structure is an expression of problem-domain complexity, pertinent to the system's responsibilities. The term "Structure" is used as an overall term, describing both Generalization- Specialization (Gen-Spec) Structure and Whole-Part Structure(34).

- association: a relationship among instances of two or more classes describing a group of links with common structure and common semantics(144).

- superkey: a set of one or more attributes which, taken collectively, allow us to identify uniquely an entity in the entity set(89).

- foreign key: a primary key of one table that is embedded in another (or the same) table.
- Inheritance: a relationship among classes, wherein one class shares the structure or behavior defined in one (single inheritance) or more (multiple inheritance) other classes(23).
- Generalization-Specialization structure: a type of relationship between classes where one class is derived from one or several general classes. Generalization-Specialization structure may be viewed as part of the "distinguishing between Classes" aspect of the three basic methods of organization(34).
- Whole-Part structure: a type of relationship between classes in which one class includes an object of another class. Whole-Part structure may be viewed as aggregation of the relationship(34).
- Object-oriented development: an approach to software design in which the decomposition of a system is based upon the concept of object.

2. Terminology of DOODBMS:
   - Distributed Database: a database that is spread over several computers of the computer network.
   - OODBMS: a database system which directly supports an object-oriented model. OODBMS must provide persistent storage for the objects and their descriptors (schema)(84).
   - Schema, scheme: the logical structure of the data; the overall design of the database. In RDBMS, the database schema is created by writing a set of definitions which are translated by the data definition language (DDL).
   - Data Item: the smallest unit of data that has meaning in the real world, such as object.
   - Record: a group of related data items.
   - Data Independence: the ability to modify a scheme definition in one level without affecting a scheme definition in the next higher level(89). Physical and logical data independence are two level of data independence.

- Logical data independence: the immunity of user applications to changes in the logical structure of the database(121).
- Physical data independence: deals with hiding the details of the storage structure from user applications. When a user application is written, it should not be concerned with the details of the physical data organization.
- Federated DBMS: a federated database system (FDBS) is a collection of cooperating but autonomous component database system (DBSs). The component DBSs are integrated to various degrees. The software that provides controlled and coordinated manipulation of the component DBSs is called federated database management system (FDBMS)(149).
- Interoperability: the ability of two or more programs to communicate or work together despite having been written in different languages or language dialects. Frequently, interoperability also implies communication between two or more different execution domains, which may range from different run- time support systems within a single processor to physically distinct processors within a distributed computing system(173).
- Multidatabase: Multidatabase systems are those where components have no notion of cooperation whereas federated systems are those that have been designed to cooperate in a limited fashion. For example, if you take Oracle and DB2 and try to build a layer of software on top of them to make them inter-operate, we call this a multidatabase system(123).
- Network transparency, distribution transparency: the capability to hide the existence of network and protect from the operational details of the network. In other words, there would be no difference between database applications that would run on a centralize database and those would run on a distributed database.
- Client/Server DBMS architectures: client/server architectures is that they are a first step toward true distribution where there is peer-to-peer communication. All the systems that are multiple client/single server are not a distributed DBMS since data is never distributed(123). A client/server sys-

tem is one in which a centralized object server manages the entire persistent database on behalf of a number of client machines. A client/server system is a restricted distributed database system. A fully distributed object-oriented database system is one in which the objects are distributed across different sites on a network, and the physical distribution of objects is completely transparent to the users of the system(84).

- Fragmentation: the data items is partitioned into several fragments. Each fragment is stored in a different site.
- Replication: the system maintains several identical replicas (copies) of the data items. Each replica is stored in a different site.
- Object Placement: the placement of object across the nodes of a computer network.
- Remote Access: Accessing data which is not stored locally.
- User view: the way in which the database is seen by the local users.

3. Terminology for Network Communications:

    - Node, site, Host: computers on a network are commonly referred to Nodes, Sites, or Hosts.
    - Centralized Computer Network: a network in which all the nodes share only the resources of a central computing node.
    - Distributed Computer Network: network in which the nodes are distributed and the computer at each node communicate over the communication subsystem. Each node can access the resources of every other node.
    - Computer Network: an interconnected collection of autonomous computers that are capable of exchanging information among them(121).

Appendix B. Data Dictionary

1. Data Dictionary for Objects&Classes

   - Academic-advisor: advise students' courses and research directions. One academic-advisor can advise more than one students.
   - Advisor: a faculty member who assume counseling a classes and thesis, such as academic advisor, thesis advisor. An advisor can have many students.
   - Classes: The courses that are offered for organization. A class may have some attend by at least one student, and teach by at least one teacher.
   - Computer-support-group: a group of people that systems installation, maintenance, etc..
   - Department: a section or division of academic organization.
   - Dissertation: a research paper, esp. one written for a doctoral degree. A dissertation can be one student or more than one student efforts.
   - Employee: a person employed by another for wages or salary. A member of employee can also be a student.
   - Employee-Student: a person who enrolls for study teaching job.
   - Faculty: persons who assume teaching job in an environment, such as professor, instructor, etc.. A member of faculty can also be a student. A teach more than one class.
   - Faculty-Student: A person who assume teaching and academic organization.
   - Generic-organization: a generic unit for an academic organization.
   - Generic-person: a generic unit for describing a information that is universal to the classes and people that are related to the problem domain.
   - Institution: an organization having a educational college. An institution can have many libraries, Computer support group etc..
   - Job-opportunity: a vacancy of job opportunity.
   - Library: a room or building, or an institution in collection of books that are related to students' such as academic library, technical library.

- Meeting: a gathering of people for specific purpose.
- Offering: courses offering in a school.
- Paper: Published papers.
- Prerequisite: the requirements of courses taken study advanced courses. A prerequisite can be required by one or more classes.
- Publication: the printing and distribution of books, thesis, dissertation, reports, etc..
- Report: a publication that are research type of work.
- Research: a systematic study and investigation in knowledge.
- SIG: (special interest group): any organization interests that are related to academic research.
- Software: this object holds information about the location of software, versions, specific project software, student's electronic version of thesis, dissertation, bulletin boards, application software, free distribution software, etc.
- Sponsor: a person(or more) or an organization student's thesis research or academic research program.
- Student: a person who is enrolled for study at a etc.. A student can also be a employee. A student than one class. A student must has at least one academic advisor, and two thesis committee members.
- Thesis: a research paper, esp. one written by a master's degree. A thesis can be one student or more than one students efforts.
- Thesis-advisor: advise students' thesis research directions. One thesis-advisor can advise more than one students. Each student is limited to have one thesis-advisor.

2. Data Dictionary for Services

   - Print-prerequisite-info: print out the prerequisite information, including course number, credit hour, comments, for taking advanced courses.

- Print-course-info: print out course information, including course number, credit hour, teacher, classroom, class hour, comments.
- Print-full-name: print out first name, middle name , and last name of a person.
- Print-organization-info: print out information including organization code, organization name, point of contact, address.
- Print-job-info: print out information including job title, job requirements, job offered by which organization.
- Print-E-mail-addr: print out the E-mail address.
- Inform-meeting-by-E-mail: this services is to send E-mail to meeting attendees 72 hours prior to the meeting.
- Print-attendees: print out information including specific meeting and its attendees.
- Print-meeting-info: print out information including meeting name, place, date, sponsor, topic-discussed, attendees, comments.
- Print-Area: Print out area-of-interest.
- Print-publication-info: print out information including publication title, author(s), date published, classification, publisher.
- Schedule-meeting : this services is to schedule meetings 2 years ahead, by entering the place, date and the name of attendees
- Print-application-info: print out information including research application name, author(s), research progress, schedule, sponsor, related research efforts.
- Print-seminars-info: print out information including seminars schedule.
- Print-class-sch: print out information including class schedule, class hours, classroom, class hour, teacher, credit hour.
- Print-related-thesis: print out information including thesis efforts that are related the selected topic. This information includes author(s), thesis title, call number.

- Print-thesis-info: print out information including thesis topic, authors, advisor(s), thesis grade.

3. Traditional data dictionary

   - SSAN = [3 numeric digits + '-' + 2 numeric digits +'-'+ 4 numeric digits]
   - LAST-NAME =[20 alphabetic digits]
   - FIRST-NAME =[20 alphabetic digits]
   - MIDDLE-NAME =[20 alphabetic digits]
   - E-MAIL-ADDR = [8 alphabetic digits +'@'+ 10 alphabetic digits]
   - SECTION-CODE = [3 alphabetic digits + 2 numeric digits]
   - POSITION-CODE =[3 alphabetic digits + 2 numeric digits]
   - RANK-CODE =[3 alphabetic digits + 2 numeric digits]
   - TITLE =[50 alphabetic digits]
   - ADDRESS =[200 alphanemuric digits]
   - JOB-TITLE = [100
   - SALARY = [6 numeric digits]
   - SUPERVISOR = [80 alphabetic digits]
   - AREA-OF INTEREST =[400 alphabetic digits]
   - GPA =[3 numeric digits]
   - MAJOR =[200 alphabetic digits]
   - ADVISOR = [80 alphabetic digits]
   - PROGRAM-CODE = [3 alphabetic digits]
   - ORGANIZATION-CODE = [7 numeric digits]
   - NAME = [80 alphabetic digits]
   - POC = [80 alphabetic digits]
   - COURSE-PROVIDED =[2000 alphabetic digits]
   - SUPPORT-AREA = [200 alphabetic digits]
   - COMPUTER-TYPE = [200 alphabetic digits]

- INSTITUTION-CODE = [7 NUMERIC DIGITS]
- AREA-OF-RESEARCH =[200 alphabetic digits]
- SPONSOR-CODE = [7 NUMERIC DIGITS]
- SPONSOR = [80 alphabetic digits]
- RESEARCH-CODE = [7 numeric digits]
- RESEARCH-NAME = [80 alphabetic digits]
- AREA-OF-REARCH = [200 alphabetic digits]
- APPLICATION-NAME = [80 alphabetic digits]
- RESEARCH-PROGRESS = [200 alphabetic digits]
- RESEARCHERS = [80 alphabetic digits]
- SPONSOR = [80 alphabetic digits]
- MEETING-CODE = [80 alphabetic digits]
- MEETING-NAME = [80 alphabetic digits]
- ATTENDEES = [2000 alphabetic digits]
- AUTHORS = [80 alphabetic digits]
- TITLE = [80 alphabetic digits]
- CLASSIFICATION = [7 NUMERIC DIGITS]
- PUBLISHER = [80 alphabetic digits]
- PUBLICATIONS = [80 alphabetic digits]
- DIRECTOR = [40 alphabetic digits]
- BELONG-TO = [80 alphabetic digits]
- JOB-OFFER-NUMBER = [7 NUMERIC DIGITS]
- JOB-TITLE = [80 alphabetic digits]
- JOB-REQUIREMENT = [200 alphabetic digits]
- OFFERED-BY = [80 alphabetic digits]
- LEADER = [40 alphabetic digits]
- BUDGET = [9 NUMERIC DIGITS]
- NUMBER-OF-ADVISEE =[3 NUMERIC DIGITS]

- NAMES-OF-ADVISEE=[400 alphabetic digits]
- COURSE-NUMBER =[4 alphabetic digits + 3 NUMERIC DIGITS]
- PREREQUISITE = [200 alphabetic digits]
- TAUGHT-BY = [80 alphabetic digits]
- CLASSROOM = [4 NUMERIC DIGITS]
- MAX-NUMBER-OF-ENROLLMENT =[3 NUMERIC DIGITS]
- COURSE-TITLE =[80 alphabetic digits]
- CLASS-HOUR = [40 alphabetic digits]
- ENROLLED-STUDENT = [400 alphabetic digits]
- CREDIT-FOUR= [2 NUMERIC DIGITS]
- RELATED-THESIS = [800 alphabetic digits]
- THESIS-TOPIC= [100 alphabetic digits]
- SPONSORED-BY = [100 alphabetic digits]
- THESIS-GRADE =[3 NUMERIC DIGITS WITH 2 DECIMAL POINTS]

Appendix C. OOA Results

This appendix lists the OOA results of the targeted system in detailed analysis using the OOATool. Figure C.1 lists the OOA results of the relationship between subject. Figure C.2 and Figure C.3 depict the OOA results on the subjects of *PEOPLE* and *ORGANIZATION* for the thesis application database. Figure C.4 shows the OOA results of the subject Publication. Figure C.5 illustrates the OOA results of the subject Meeting. Figure C.6 explains the OOA results of the subject Advisor. Figure C.7 lists the OOA results of the subject Student. Figure C.10 relates the OOA results of the instance connection of Advisor and Student. Figure C.8 depicts the OOA results of the subject Project. Figure C.9 lists the OOA results of the subject Course.

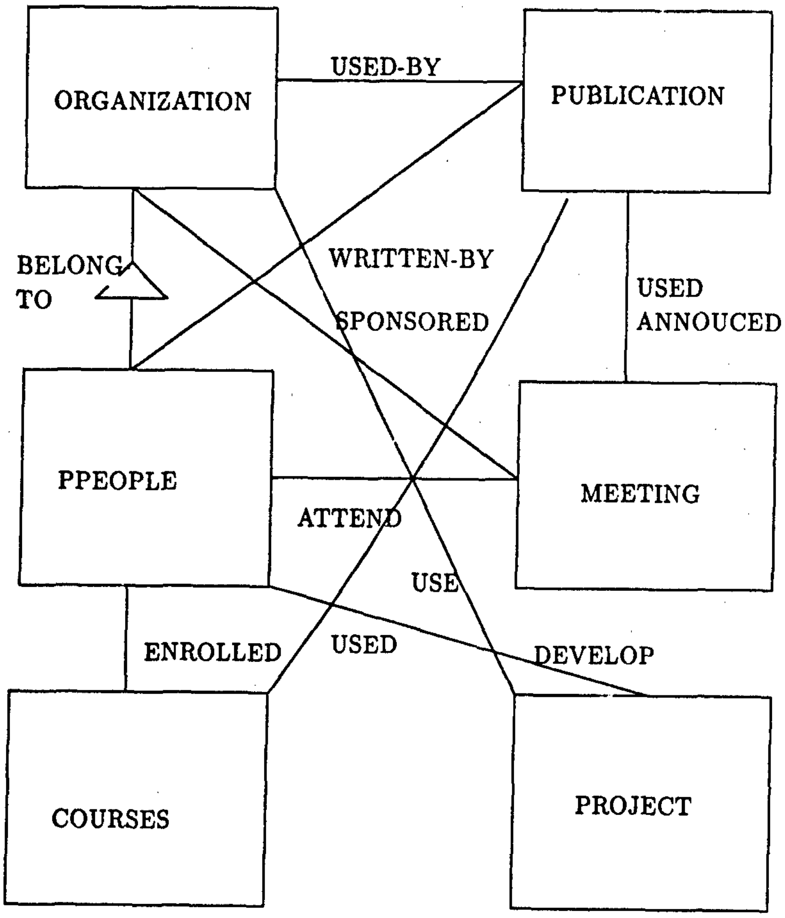


Figure C.1. OOA Results of the Relationships between Subjects

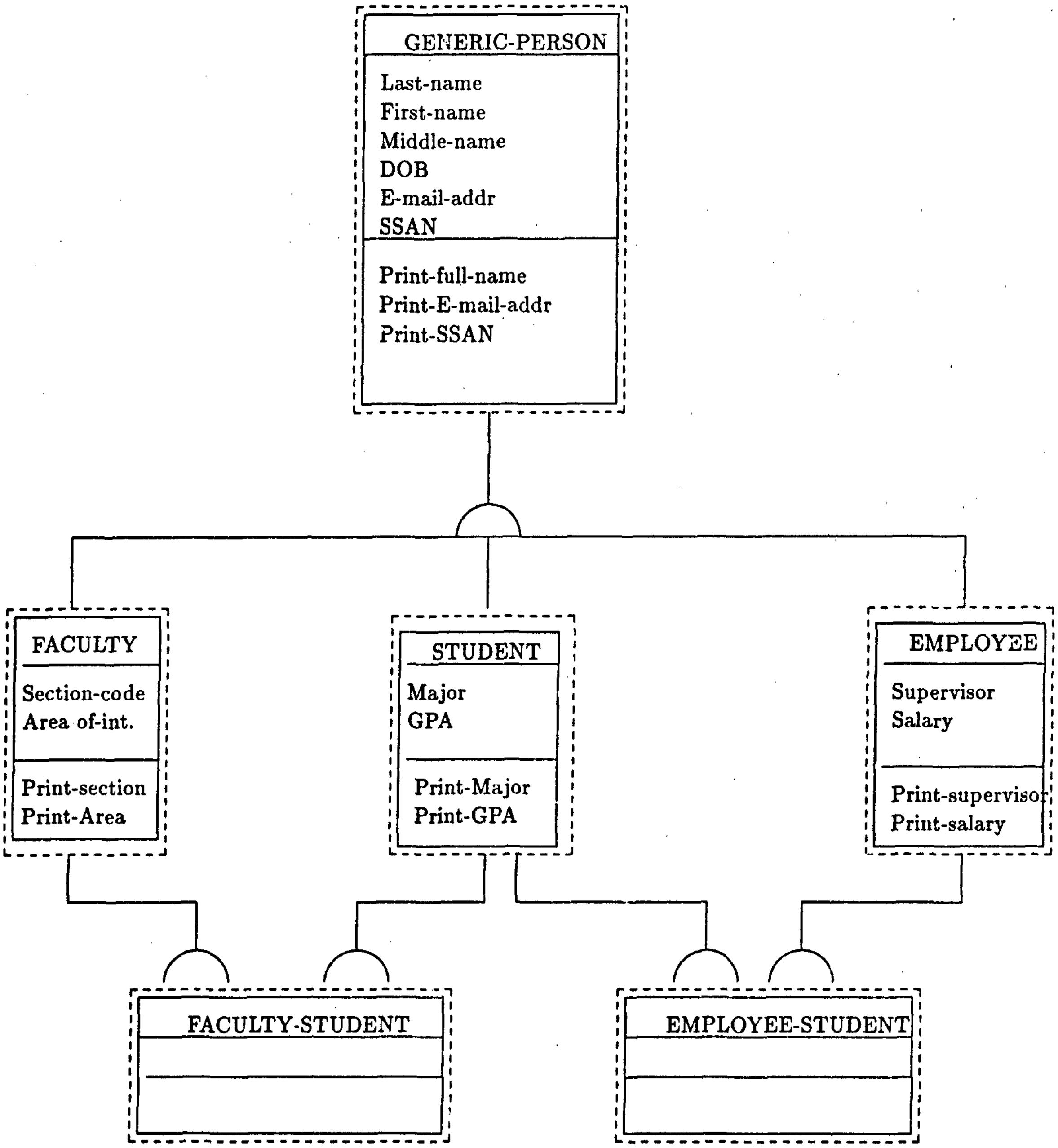


Figure C.2. OOA Results of the Subject PEOPLE

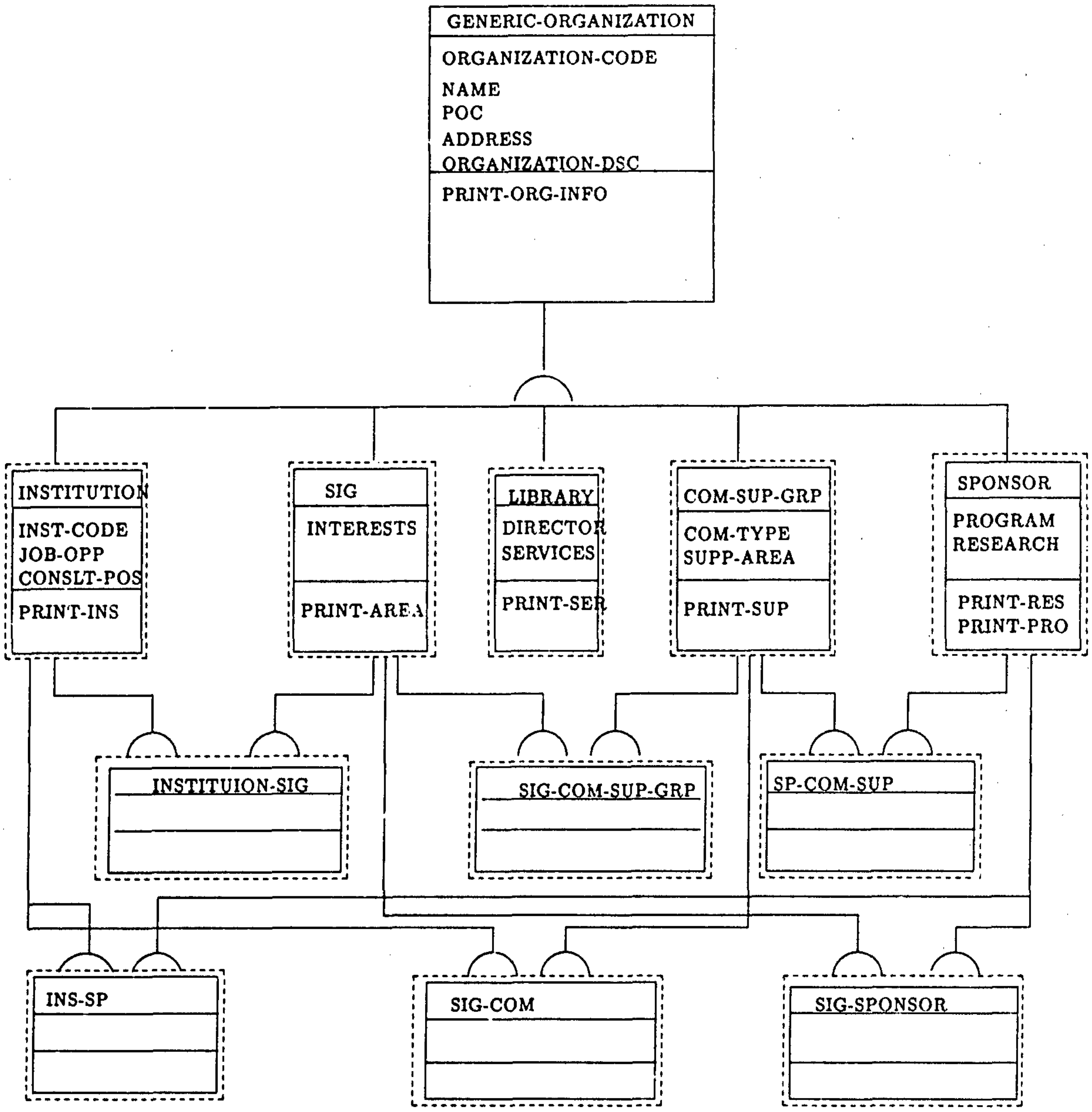


Figure C.3. OOA Results of the Subject ORGANIZATION

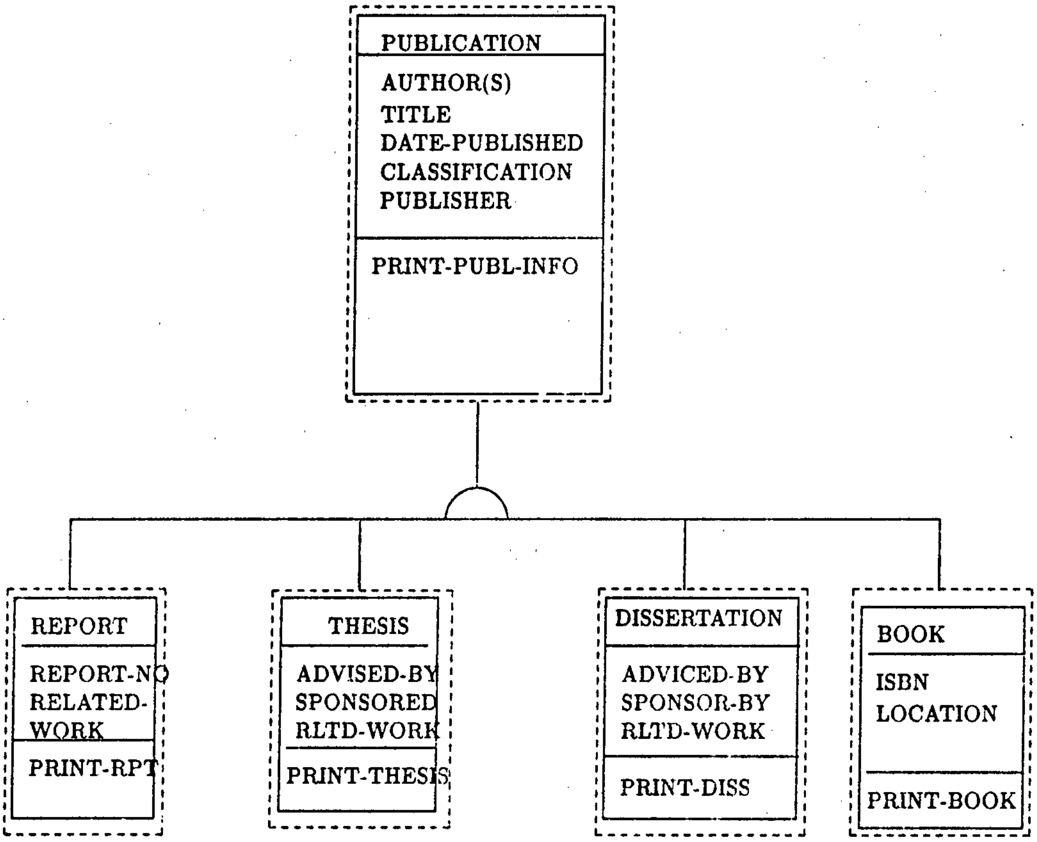


Figure C.4. OOA Results of the Subject PUBLICATIONS

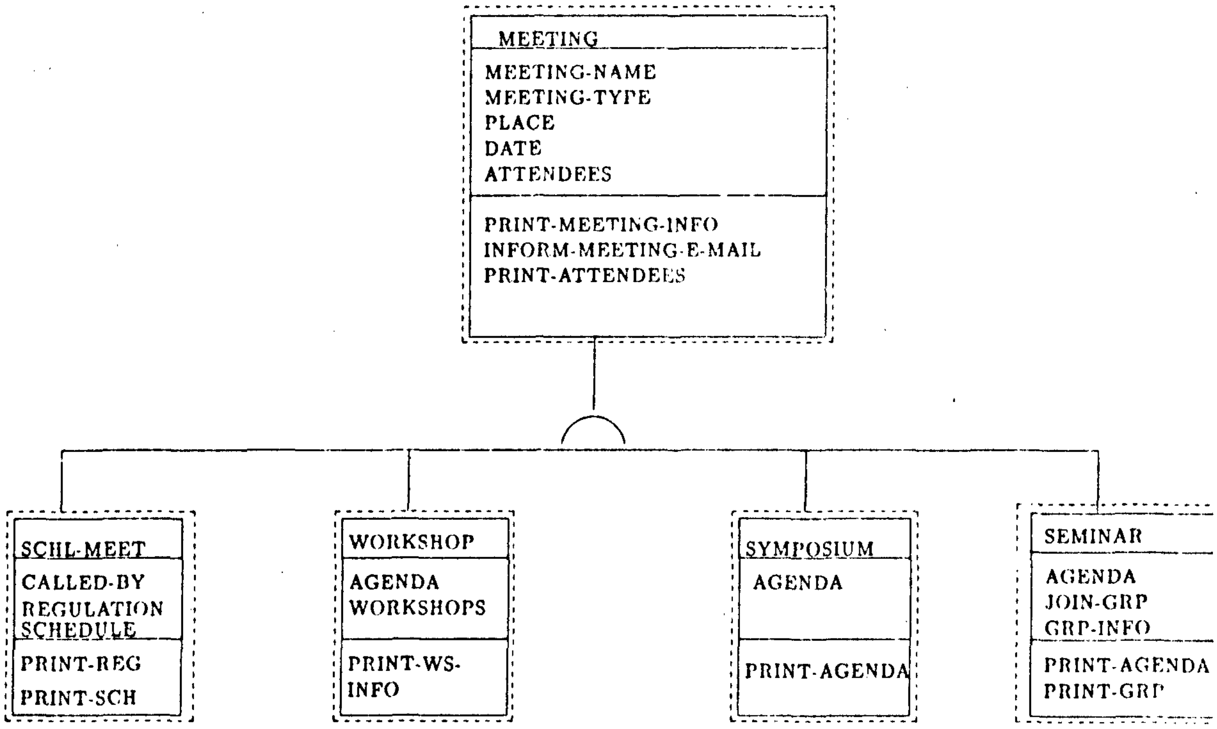


Figure C.5. OOA Results of the Subject MEETING

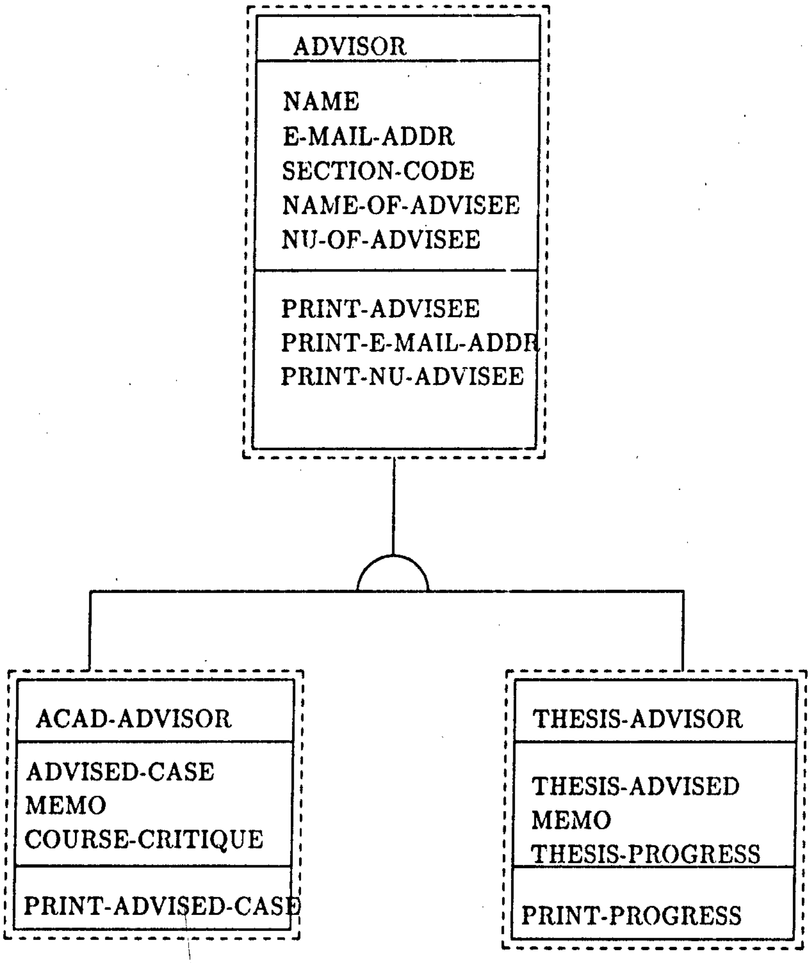


Figure C.6. OOA Results of the Subject ADVISOR

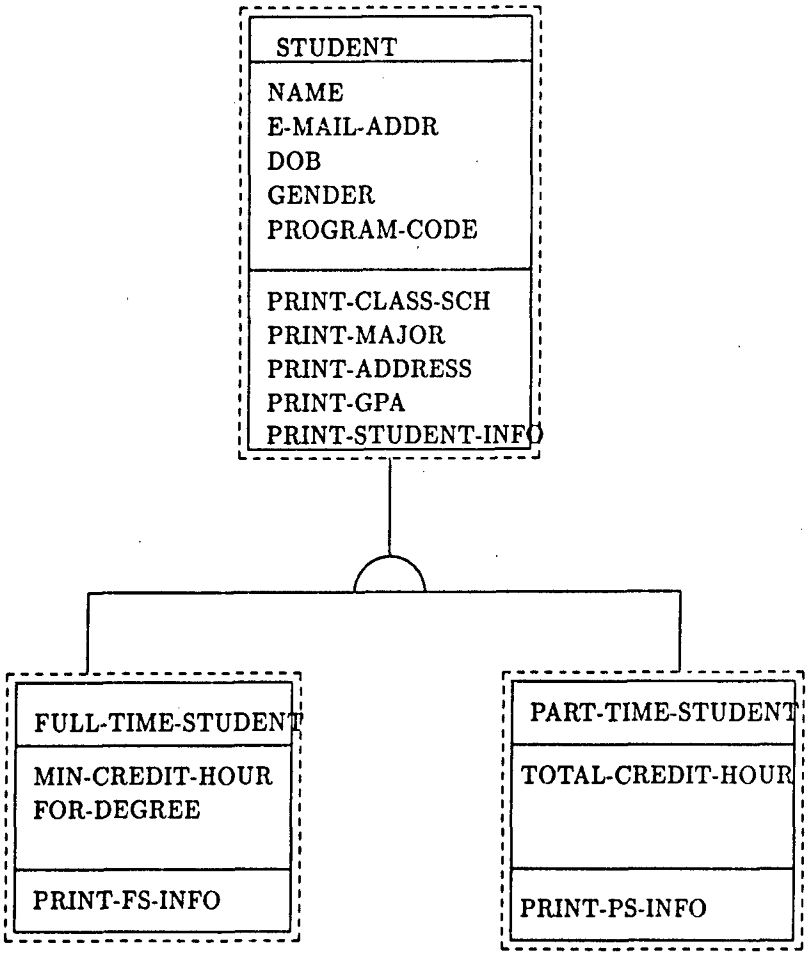


Figure C.7. OOA Results of the Subject STUDENT

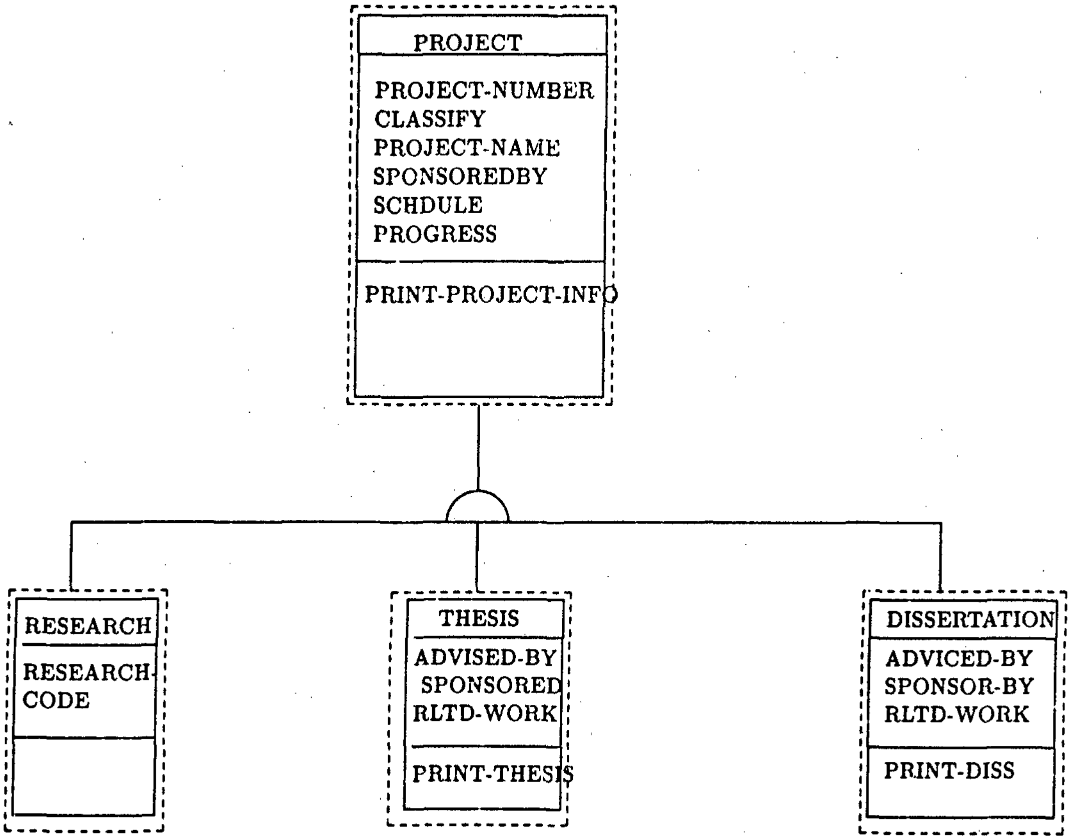


Figure C.8. OOA Results of the Subject PROJECT

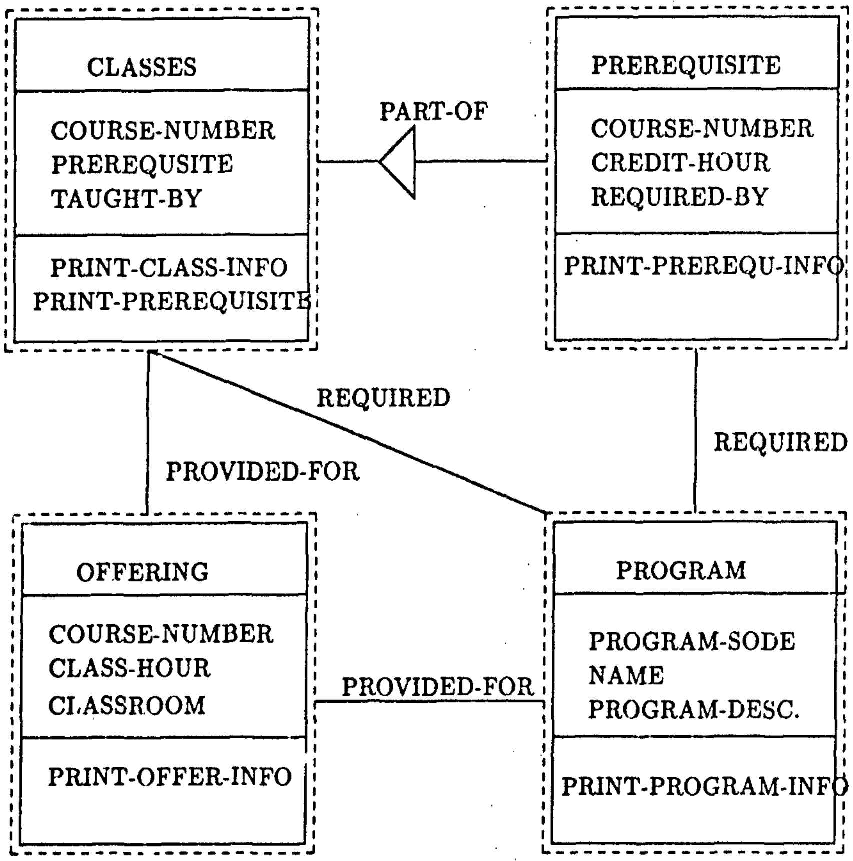


Figure C.9. OOA Results of the Subject COURSE

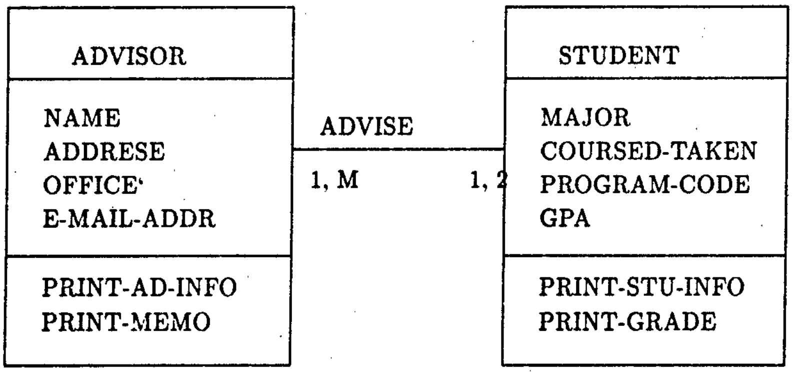


Figure C.10. Instance Connection of the Advisor and STUDENT

Appendix D. OOD Results

This appendix lists the OOD results of the targeted system in the design phase by identifying subclasses and superclasses.

1. Identify Subclasses and Superclasses: For each class and object defined in the application, the following lists all the subclasses and superclasses:

   – Identify Subclasses :

     * Advisor: Academic-advisor, thesis advisor.
     * Book : (none)
     * Classes: (none)
     * Computer support group :(none)
     * Conference :(none)
     * Department: (none)
     * Dissertation:(none)
     * Employee: Employee-student.
     * Employee-Student:(none)
     * Faculty: advisor, academic-advisor, faculty-student, thesis-advisor.
     * Faculty-Student: generic-person, faculty, student.
     * Full-time-student : generic-person, student.
     * Generic-organization: Computer support group, department, instution, library, SIG, sponsor.
     * Generic-person: academic-advisor, Advisor, Employee, Employee-Student, Faculty, Faculty-Student, Full-time-student, Part-time-student, student, thesis-advisor.
     * Grade-type: (none)
     * Institution: generic-organization.
     * Job-opportunity:(none)
     * Library: generic-organization.

- Meetings: Conference, school-meeting, seminars, symposiums, workshop-meeting.
- Offering : (none)
- Papers: Publication.
- Part-time-student : generic-person, student.
- Prerequisite : (none)
- Program : (none)
- Project : Dissertation, research, thesis.
- Publications : Book, dissertation, papers, report, thesis.
- Reports:(none)
- Research : project.
- School-meeting : meeting.
- Seminar : meeting.
- Special Interest Groups(SIG) : generic-organization.
- Software : (none)
- Sponsor : generic-organization or generic-person.
- Student: Employee-Student, Faculty-Student, Full-time-student, Part-time-student.
- Symposiums :(none)
- Thesis: (none)
- Thesis-advisor :(none)
- Workshop-meeting :(none)

– Identify Superclasses:

- Academic-Advisor: generic-person, faculty, and advisor.
- Advisor: generic-person, faculty.
- Book : Publication.
- Classes: (none)
- Computer support group : generic-organization.

* Conference : meeting.
* Department: generic-organization.
* Dissertation: Publication, project.
* Employee: generic-person.
* Employee-Student: generic-person, employee, student.
* Faculty: generic-person.
* Faculty-Student: generic-person, faculty, student.
* Full-time-student : generic-person, student.
* Generic-organization: (none)
* Generic-person: (none)
* Grade-type: (none)
* Institution: generic-organization.
* Job-opportunity:(none)
* Library: generic-organization.
* Meetings: (none)
* Offering : (none)
* Papers: Publication.
* Part-time-student : generic-person, student.
* Prerequisite : (none)
* Program : (none)
* Project : (none)
* Publications : (none)
* Reports: Publication.
* Research : project.
* School-meeting : meeting.
* Seminar : meeting.
* Special Interest Groups(SIG) : generic-organization.
* Software : (none)

- Sponsor : generic-organization or generic-person.
- Student: generic-person.
- Symposiums : meeting.
- Thesis: Publication, project.
- Thesis-advisor : advisor, faculty, generic-person.
- Workshop-meeting : meeting.

Appendix E. OOA Classes in the Problem Domain

The boundary of the problem domain of the targeted system can be defined by performing OOA. This distributed DBMS application maintains the following information:

- Academic-Advisor: Last-Name, First-name, middle-name, DOB, mail and E-mail address, Work-phone, Home-phone, area-of-interest, Section-code, Number-of-advisee, name-of-advisee. The services of this object are: Print-advisees, print-area, print-phone, print-address, print-e-mail-addr, print-full-name.
- Advisor: (thesis-advisoı, academic-advisor) Last-Name, First-name, middle-name, DOB, mail and E-mail address, Work-phone, Home-phone, area-of-interest, Section-code. The services of this object are: print-address, print-e-mail-addr, print-full-name.
- Book : Author(s), Title, date-published, classification, publisher. The services of this object are: Print-publication-info.
- Classes: courses-number, prerequisite, taught-by, classroom, max-number-of-enrollment, course-title, class-hour, enrolled-students, credit-hour. The services of this object are: Print-prerequisite, Print-course-info.
- Computer support group : Organization-code, name of company, POC, computer-types, support area, mail and e-mail addresses. The services of this object are: Print-organization-code, Print-group-info, print-poc, Print-Name.
- Conference : Conference-number, Meeting-code, meeting-name, Meeting-type, Place, Date, Attendees, Sponsored-by. (for scheduling conferences, workshops, symposiums 2 years ahead). The services of this object are: Inform-meeting-by-e-mail, print-attendees, print-meeting-info.
- Department: Section code, chief, POC, name, Organization code, budget. The services of this object are: Print-organization-code, Print-group-info, print-poc, Print-Name, Print-budget.

- Dissertation: dissertation-topic, author(s), related-work, followed-up-work, Date-published, advised-by, sponsored-by, dissertation-grade. The services of this object are: Print-publication-info.
- Electronic-Document: Document No., Title, Authors, addresses(location of) , specific project documents, free distribution documents, etc.. The services of this object are: Print-document-info.
- Employee: Last-Name, First-name, middle-name, DOB, mail and E-mail address, Work-phone, Home-phone, Supervisor, salary. The services of this object are: print-address, print-e-mail-addr, print-full-name, Print-supervisor, print-phone.
- Employee-Student: Last-Name, First-name, middle-name, DOB, mail and E-mail address, institution, interest-area, Major, Class-taken, gpa, home-phone, Supervisor, salary. The services of this object are: print-address, print-e-mail-addr, print-full-name Print-class-sch, Print-GPA, Print-major.
- Faculty: Last-Name, First-name, middle-name, DOB, mail and E-mail address, position, area-of-interest, Section-code. The services of this object are: print-address, print-e-mail-addr, print-full-name, Print-area, Print-phone.
- Faculty-Student: Last-Name, First-name, middle-name, DOB, mail and E-mail address, position, area-of-interest, Section-code, institution, Major, GPA, Class-taken, interest-area, home-phone. The services of this object are: print-address, print-e-mail-addr, print-full-name, Print-area, Print-phone, Print-class-sch, Print-gpa, Print-major.
- Full-time-student : Last-Name, First-name, middle-name, DOB, gender, program-code, mail and E-mail address, institution, interest-area, Major, Class taken, Minimum-credit-hour, gpa, home-phone. The services of this object are: Print-class-sch, Print-GPA, Print-major, print-phone, Print-address, Print-e-mail, Print-full-name.
- Generic-organization: Organization-code, Name, POC, Organization description. The services of this object are: Print-organization-info, Print-POC.

- Generic-person: SSAN, Last-Name, First-name, middle-name, DOB, gender, mail and E-mail address, Work-phone, Home-phone. The services of this object are: print-phone, Print-address, Print-E-mail-addr, Print-full-name.
- Grade-type – latter-grade, grade-point. The services of this object is: print-grade-type.
- Institution: Organization-code, name, Address, Organization-description, POC, job possibilities, consulting possibilities. The services of this object are: Print-organization-info, Print-POC.
- Job-opportunity: Job title, job-requirement, offered-by. The services of this object are: Print-job-info.
- Library: directories, Name, Belong-to, POC, mail and e-mail addresses, sponsored-by, director, librarian). The services of this object are: Print-organization-info, Print-POC.
- Meetings: Meeting-code, meeting-name, Meeting-type, Place, Date, Attendees, Sponsored-by. (for scheduling conferences, workshops, symposiums 2 years ahead). The services of this object are: Inform-meeting-by-e-mail, print-attendees, print-meeting-info.
- Offering : course-number, classroom, class hour, taught-by, Prerequisites, max-enrollment, offered-quarter. The services of this object is: Print-course-info.
- Papers: Title, author(s), Classification, related-work, followed-up-work. Date-published, advised-by, sponsored-by. The services of this object is: Print-publication-info.
- Part-time-student : Last-Name, First-name, middle-name, DOB, gender, program-code, mail and E-mail address, institution, interest-area, Major, Class-taken, gpa, home-phone. The services of this object are: Print-class-sch, Print-GPA, Print-major, print-phone, Print-address, Print-e-mail, Print-full-name.
- Prerequisite : Course number, credit hour. prerequisite. The services of this object is: Print-prerequisite-info.

- Program : program name, program code, program description, program comment, program requirement, number-of-student. The services of this object is: Print-program-info.

- Project : Project number, classify, expenditure, sponsoredby, project name, project developer(s), project schedule, project progress. The services of this object is: Print-project-info.

- Publications : Author(s), Title, Date-Published, Classification, Publisher(for theses, dissertations, papers, reports, etc..). The services of this object is: Print-publication-info.

- Reports: Report-number, Title, author(s), Date-published, publisher, classification. The services of this object is: Print-publication-info.

- Research : Research code, application name, author(s), research progress, sponsored-by, research schedule. (for government sponsored, industry sponsored, application procedures, etc..). The services of this object are: Print-application-info, Print-schedule, Print-progress.

- School-meeting : Called-by, Meeting-code, meeting-name, Meeting-type, Place, Date, Attendees, Sponsored-by. (for scheduling conferences, workshops, symposiums 2 years ahead). The services of this object are: Inform-meeting-by-e-mail, print-attendees, print-meeting-info.

- Seminar : Meeting-code, meeting-name, Meeting-type, Place, Date, Attendees, Sponsored-by. (for scheduling conferences, workshops, symposiums 2 years ahead). The services of this object are: Inform-meeting-by-e-mail, print-attendees, print-meeting-info, Print-seminar-info.

- Special Interest Groups(SIG) : Organization-code, Organization-name, Organization-description, Meeting-schedule, seminars, members, mail and e-mail addresses, POC, specific interests, specific information(i.e., Education SIG, text books, courses, course material, seminars ...). The services of this object are: Print-seminars-info, Print-meeting-sch, print-organization-info, Print-POC, Print-name.

- Software : Software No., Title, Authors, addresses(location of) , specific project software, bulletin boards, application software, free distribution software, etc.. The services of this object are: Print-bulletin-info, Print-software-info, Print-location.

- Sponsor : Organization code, Name, POC, Area-interest(Sponsor can be a single person, more than one person, or an organization), Organization-description. The services of this object are: Print-organization-info, Print-POC, Print-name.

- Student: Last-Name, First-name, middle-name, DOB, gender, program-code mail and E-mail address, institution, interest-area, Major, Class-taken, gpa, home-phone. The services of this object are: Print-class-sch, Print-GPA, Print-major, print-phone, Print-address, Print-e-mail, Print-full-name.

- Symposiums : Meeting-code, meeting-name, Meeting-type, Agenda, Place, Date, Attendees, Sponsored-by. (for scheduling conferences, workshops, symposiums 2 years ahead). The services of this object are: Inform-meeting-by-e-mail, print-attendees, print-meeting-info, Print-symposiums-info. Print-agenda.

- Thesis: thesis-topic, author(s), Classification, Publisher, related-thesis, followed-up-thesis, Date-published, advised-by, sponsored-by, thesis-grade. The services of this object are: Print-related-thesis, Print-thesis-info, print-publication-info, Print-sponsor-info.

- Thesis-advisor : Last-Name, First-name, middle-name, DOB, mail and E-mail address, area-of-interest, Section-code, Number-of-advisee, name-of-advisee, Thesis-topics. The services of this object are: Print-advisees-info, Print-thesis-topic, print-area, print-phone, print-address, print-e-mail addr, print-full-name.

- Workshop-meeting : Participated-workshop, Meeting-code, meeting-name, Meeting-type, Place, Date, Attendees, Sponsored-by, Agenda. (for scheduling conferences, workshops, symposiums 2 years ahead). The services of this object are: Inform-meeting-by-e-mail, print-attendees, Print-workshop-info, print-meeting-info, Print-agenda.

The services of this distributed database include the following:

- retrieve and update information in the database,
- modify data at the local site,
- look up information of publications of libraries,
- keep track of related and follow-up thesis work,
- locate software, electronic copy of thesis, document etc..
- report forms of student, employee, faculty's related information,
- print forms of employee's related information.
- print forms of research information,
- print forms of thesis related information,
- print forms of publication information,
- print forms of organization information,
- generate course information,
- generate class schedule for student,
- generate class roster for faculty members,
- print class offering information,
- contact support groups for meeting or research conference,
- schedule meetings information for conference or workshop,
- send meeting reminders to meeting attendees of faculty, student and support personnel, etc.

Appendix F. An Application's Connections to the ITASCA Platform

This appendix illustrates the application's connections to the ITASCA platform. These connections are essential because the application relies on the DOO-DBMS functions provided by the platform. The following items explain how the application program is implemented with the aids of the ITASCA C++ API:

- To connect to the server: this is the first step that the C++ API programs can utilize the functions. To connect:

  ```
        int status;
  status = ITASCA::promptConnect("hawkeye", "itasca-interface");
  ```

  If the connection is established, ITASCA return a *status* integer number that is greater than 1.

- Writing Data to Instance of Classes in ITASCA: After classes&Objects are created using DSE Tools, object instances of a given class are created in the C++ programs. The header files define the persistent classes which defined in the DSE and the application program. A complex composition of class is accomplished by assigning the object as a pointer to an attributes.

  ```
  GenericPerson genericperson1;
  genericperson1.firstName = &inputattribute1;
  genericperson1.lastName  = &inputattribute2;
  genericperson1.ssn       = &inputattribute3;

  GenericOrganization genericorganization1;
  genericorganization1.organizationName = &inputattribute1;
  genericorganization1.organizationPoc  =  &genericperson2;
  genericorganization1.organizationDes    =  &inputattribute2;
  ```

- Query Functions: Even the query used in the object-oriented DBMS is non-navigational, non-professional users still find it bothersome to retrieve data. To solved this problem, a X Window graphical user interface query is proposed. To query the application, the following functions is utilized:

  ```
  Iselect_any_star(&uid, &inputClassName,
        QUERY_EXPRESSION,
       "(and (equal &input1 \&instance1 \") \
  (equal &input2 \&instance2
        \"))",END_ARGS);
  ```

  The fact that the use of graphical user interface can further hide the query languages into graphical representations or prompts. In this way, users are not encumbered by the learning curve of the query language constructs.

- Schema Functions: the following schema functions are applied in the application to facilitate the change of schema in the application dynamically(such as users' inputs through a graphical user interface):

```
        attr = IUmake_attribute(&input1,
                 DOMAIN, "string", END_ARGS);
         attr2 = IUmake_attribute(&input2,
                 DOMAIN, "string", END_ARGS);
         IUappend(attr, attr2);
          attr2 = IUmake_attribute(&input3,
                 DOMAIN, "string", END_ARGS);

        Imake_class(&uid, &inputClassName,
                 ATTRIBUTES, attr, END_ARGS);
         IUfree_generic(attr);
```

  The instances can also be created dynamically applying the ITASCA functions to the application:

```
 Imake(&uid2, , &inputClassName
      ":&input1 \&instance1 \
       :&input2 \&instance2 \
       :&input3 \&instance3 \
       :&inputn \&instancen");
```

- Creating, modifying and deleting classes or superclasses: the application should be allowed users creating, modifying and deleting classes dynamically(such as users' inputs through a graphical user interface). This is achieved through the following code implementation:

```
status = ItascaClassObject::makeClass( &inputClassName,
          "CLASS", FALSE,
         INTLIST, "(SET-OF INTEGER)", END_ARGS);

status = ItascaClassObject::renameClass( &inputClassName,
          "CLASS", FALSE,
         INTLIST, "(SET-OF INTEGER)", END_ARGS);

status = ItascaClassObject::deleteObject( &inputClassName,
          "CLASS", FALSE,
         INTLIST, "(SET-OF INTEGER)", END_ARGS);

status = ItascaClassObject::addSuperclass( &inputClassName,
          "CLASS", FALSE,
         INTLIST, "(SET-OF INTEGER)", END_ARGS);

status = ItascaClassObject::removeSuperclass( &inputClassName,
          "CLASS", FALSE,
         INTLIST, "(SET-OF INTEGER)", END_ARGS);

Isupperclasses(&class_list, &inputClassName);
for (nlp = class_list;  nlp;  nlp = nlp->next)
                printf("Superclass(es)  are(is) %s\n", nlp->val);

Isubclasses( &class_list, &inputClassName );
for ( nlp = class_list;  nlp;  nlp = nlp-> next)
  printf("Subclass(es)  are(is) %s\n", nlp-> val);
```

- Using versioned objects: versioning allows users to track versions of an objects(such as algorithms implementations into same language but different revision). To take advantage of OODBMS and ITASCA functions, the following versioned controls are implemented:

```
        Imake_class(&uid, &inputClassName ,
         ATTRIBUTES, attr,
         VERSIONABLE, TRUE, END_ARGS);
         IUfree_generic(attr);

         Iversion_number(&version_number, version_instance);
         printf("Version number  is %s\n", version_number);
```

- Change notification:in a distributed environment, changes can made consistenly(approved by the global database) or inconsistenly(not yet approved). To inform users when a change has occurred, the following change notification are implemented:

```
Imake_class(&uid, &inputClassName,
 NOTIFY, TRUE, END_ARGS);
```

- Database Utilities: Commit, Abort, and Disconnect.

  After completing a transaction, Commit, Abort, or Disconnect is executed to either preserve the data or discard.

```
(void) ITASCA::commit();
        (void) ITASCA::disconnect();
```

The prototype source code is located in hawkeye server under the sub-directory *hwu/api.*

Appendix G. Depiction of the Function Calls

This appendix depicts the function calls of the application. In the process of function calls' depiction (relating to user interface functions) of the application, three functions are specified: 1) sequential functions, 2) hierarchical functions, 3) functions calling other functions. Figure G.1 shows the functions calls relating to the user interface functions. Figure G.2 depicts the sequence of operations for *CREATE.* Figure G.3 illustrates the sequence of operations for *QUERY.* Figure G.4 relates the sequence of operations for *UPDATE.* Figure G.5 shows the sequence of operations for *PRINT.* Figure G.6 explains the sequence of operations for *HELP.*

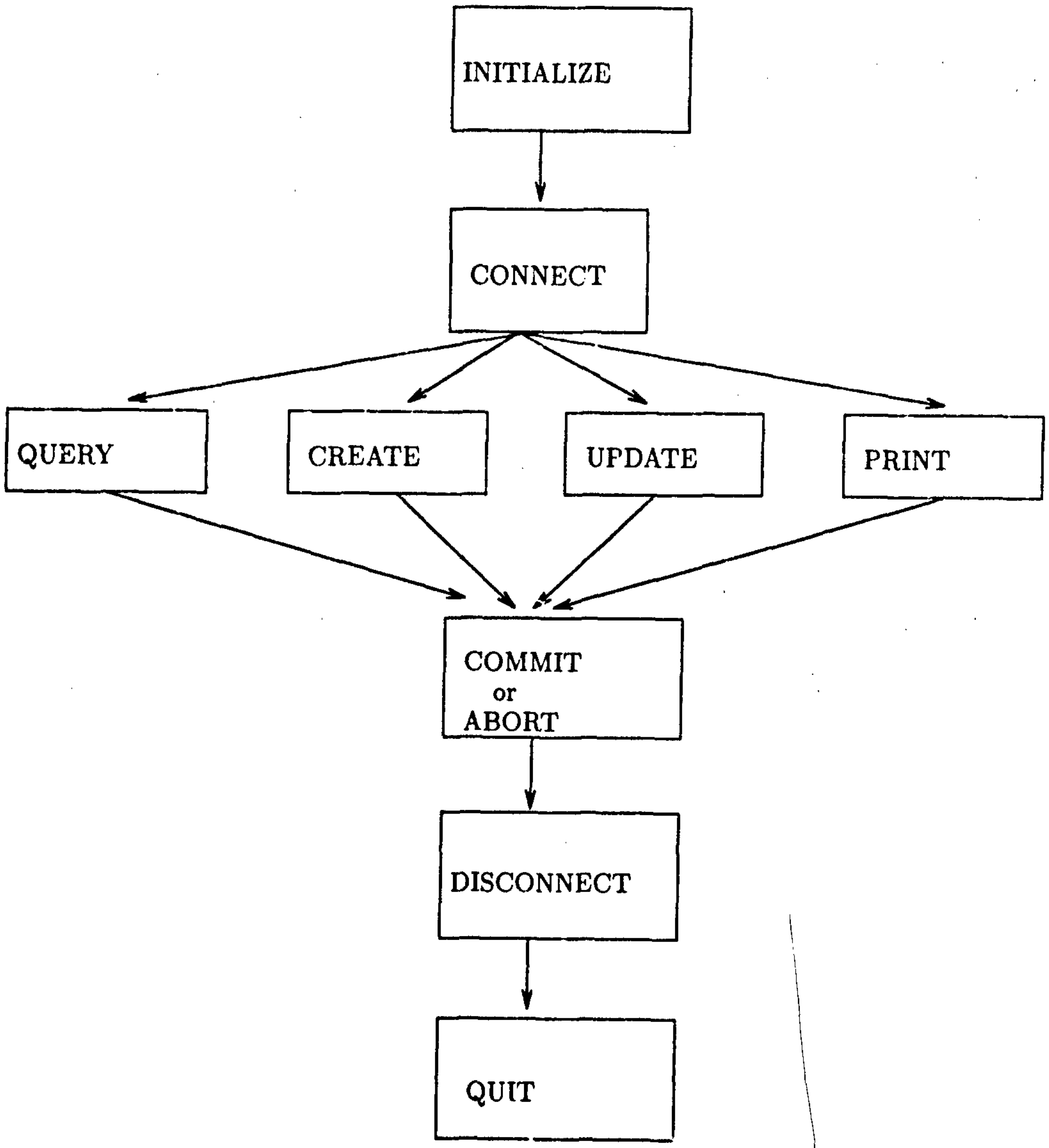


Figure G.1. Function Calls' Depiction of the PAGG DOODBMS Application

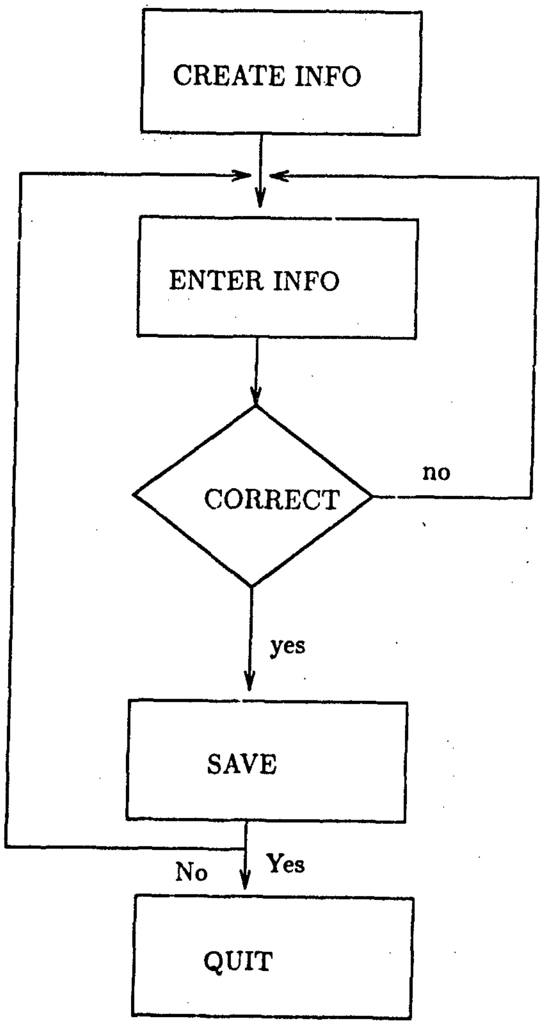


Figure G.2. Function Calls' Depiction of the Top-level Menu: CREATE

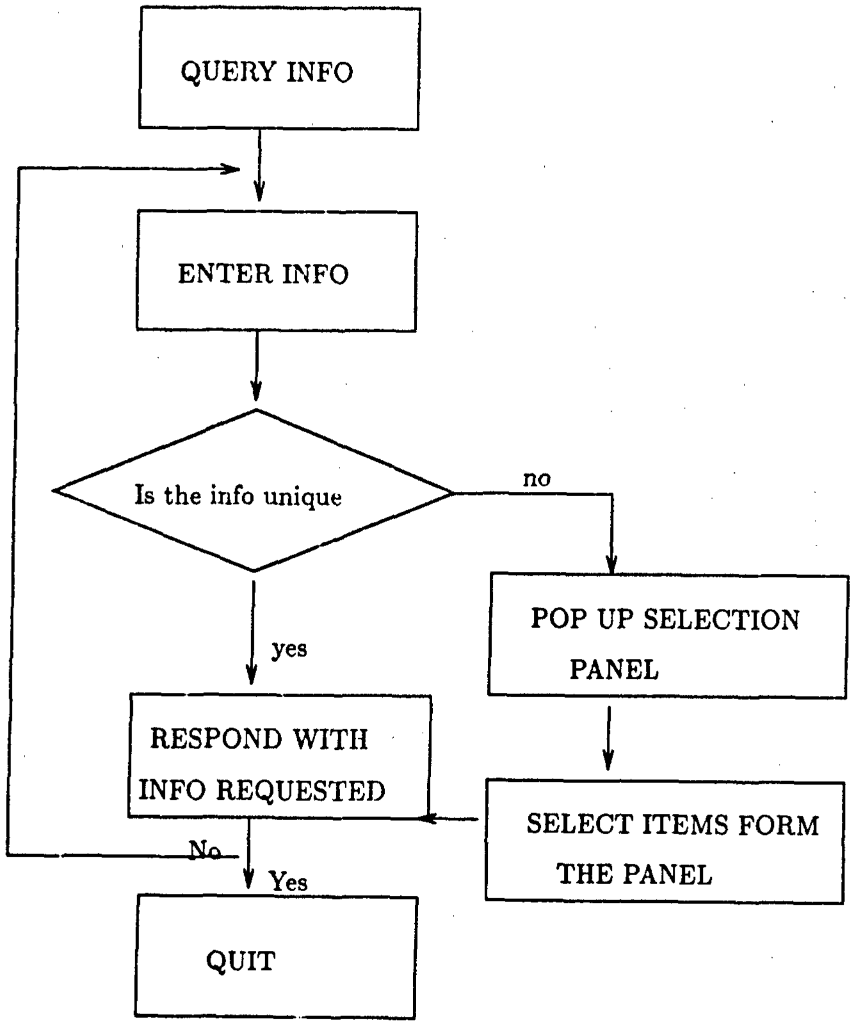


Figure G.3. Function C.'ls' Depiction of the Top-level Menu: QUERY

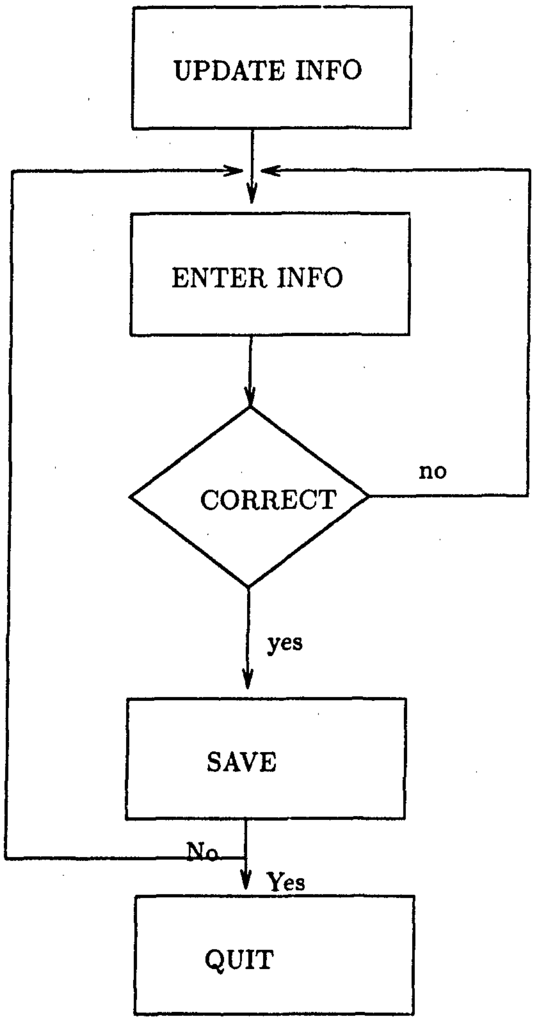


Figure G.4. Function Calls' Depiction of the Top-level Menu: UPDATE

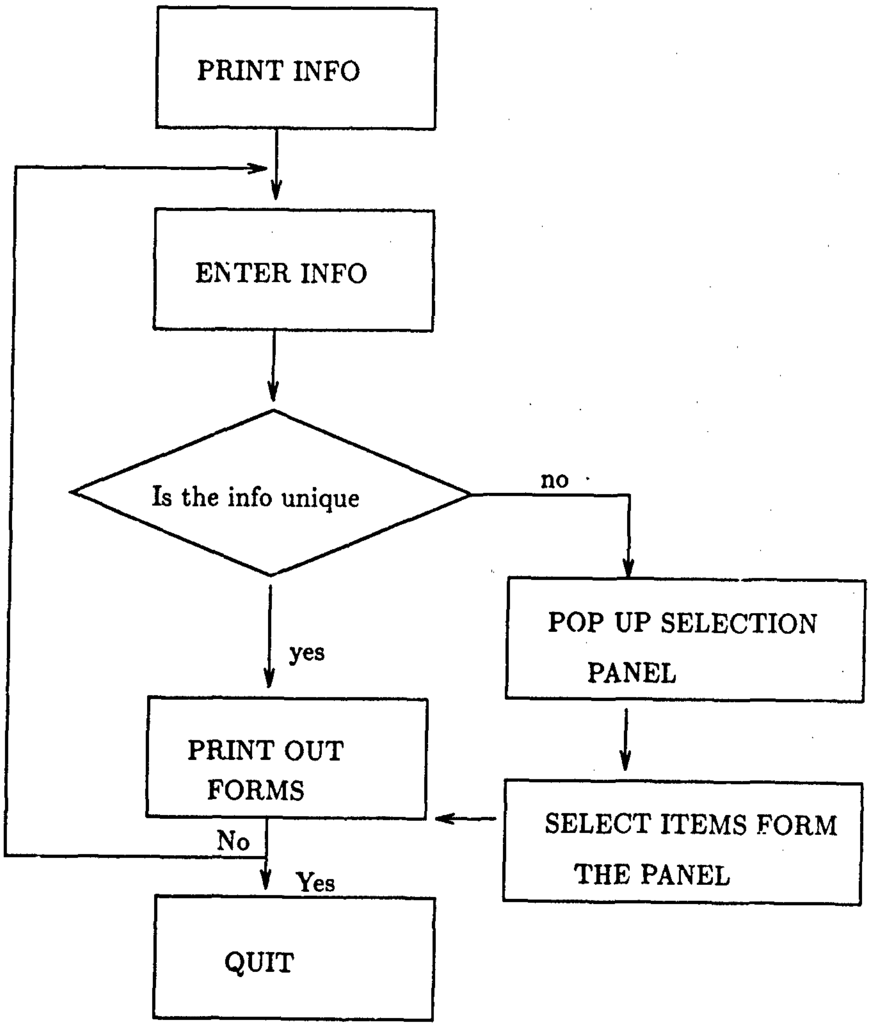


Figure G.5. Function Calls' Depiction of the Top-level Menu: PRINT

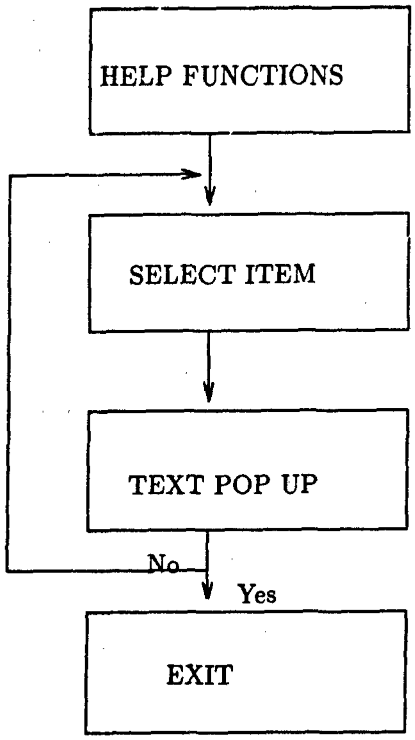


Figure G.6. Function Calls' Depiction of the Top-level Menu: HELP

Appendix H. User Interface Preliminary Results

This appendix lists the preliminary results of the user interface prototype. This user interface is intended to hide query language constructs into graphical user interface representations so that the database application could be more easily and efficiently accessed. Figure H.1 and Figure H.2 show examples of decomposing highest-level task to the second highest level as a pull-down menu. Figure H.3, H.4, H.5, H.6, H.7, H.8, illustrate pop-up windows for creating information. The screen forms of a pull-down menu button allows users to type in data, or record of data to achieve the purpose of higher level tasks. The screen form, together with some notifiers(such as OK, Cancel, and Quit) are the lowest level tasks. Figure H.9 is an example of pop-up windows for querying information. Figure H.10 shows an example of pop-up windows for updating information. Figure H.11 is a preliminary result of *PRINT* utilities. Figure H.12 illustrates a file selection panel of the application user interface. The source codes of detail implementation are located in the hawkeye file server under *hwu/PAAG* sub-directory.

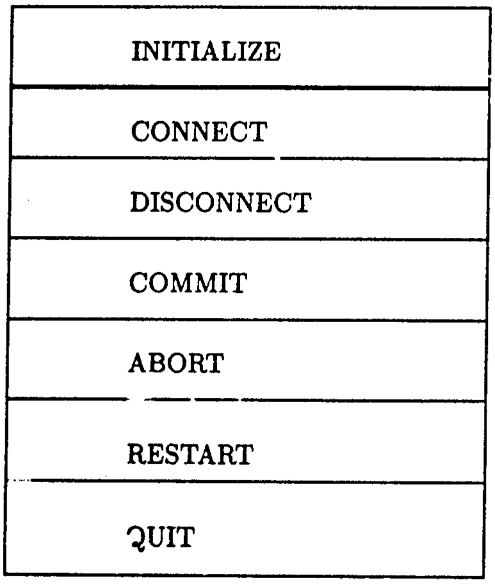


Figure H.1. Decomposing Highest-level Tasks: DB Utilities

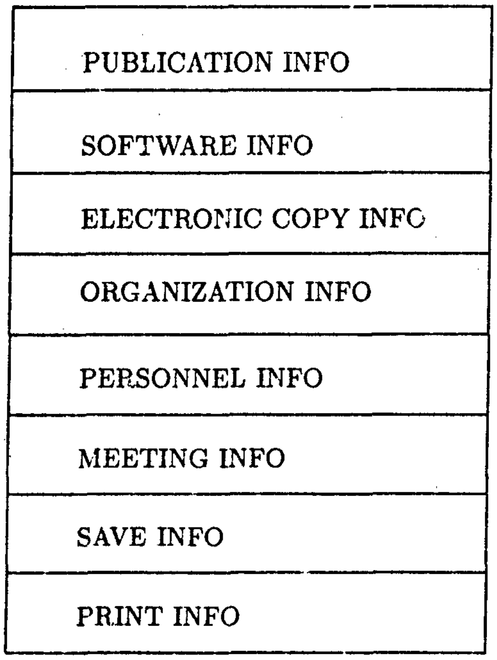


Figure H.2. Decomposing Highest-level Tasks: CREATE, UPDATE, QUERY and PRINT

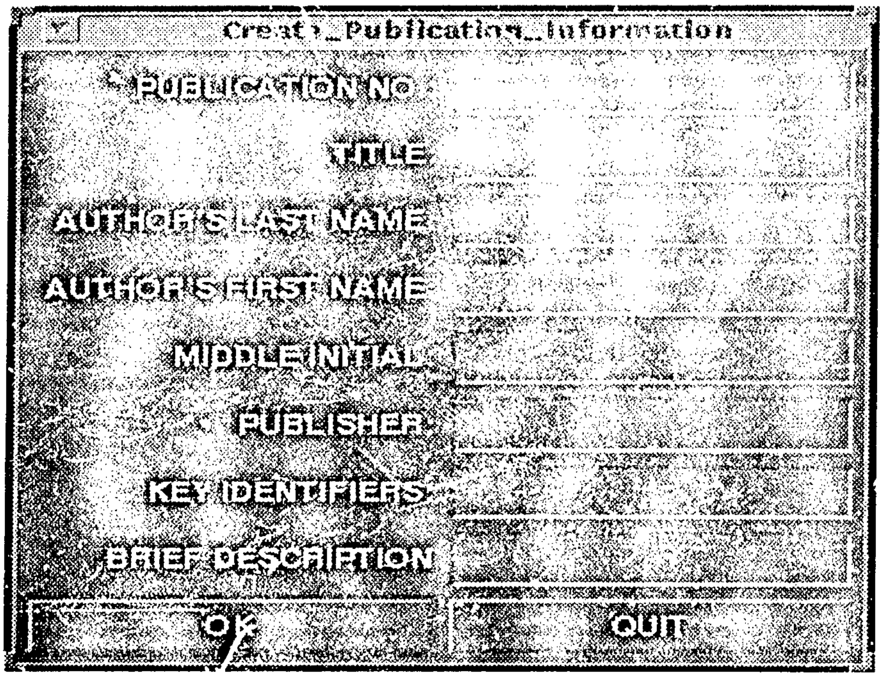


Figure H.3. Creating Publication Information Popup

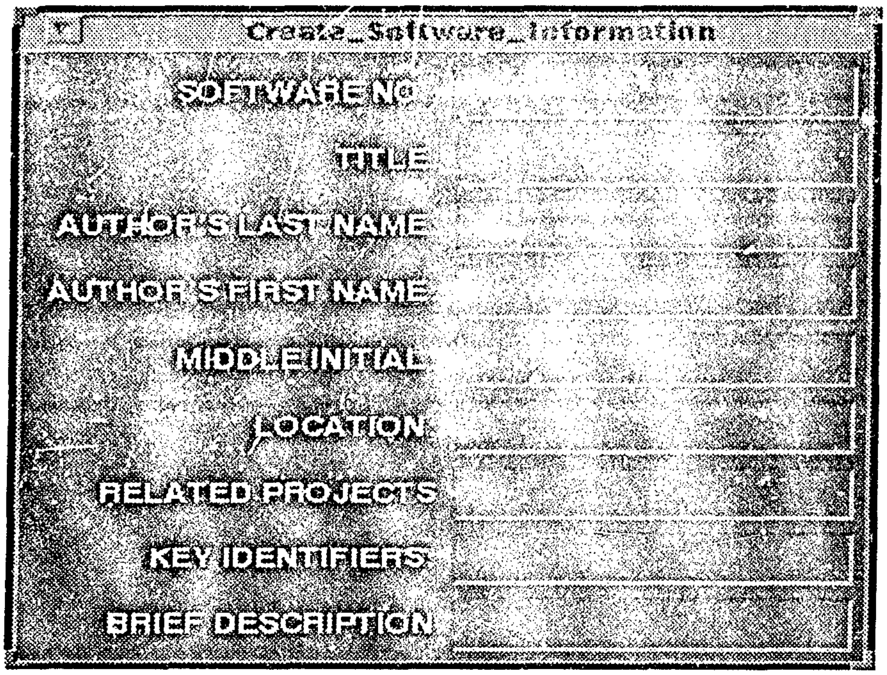


Figure H.4. Creating Software Information Popup

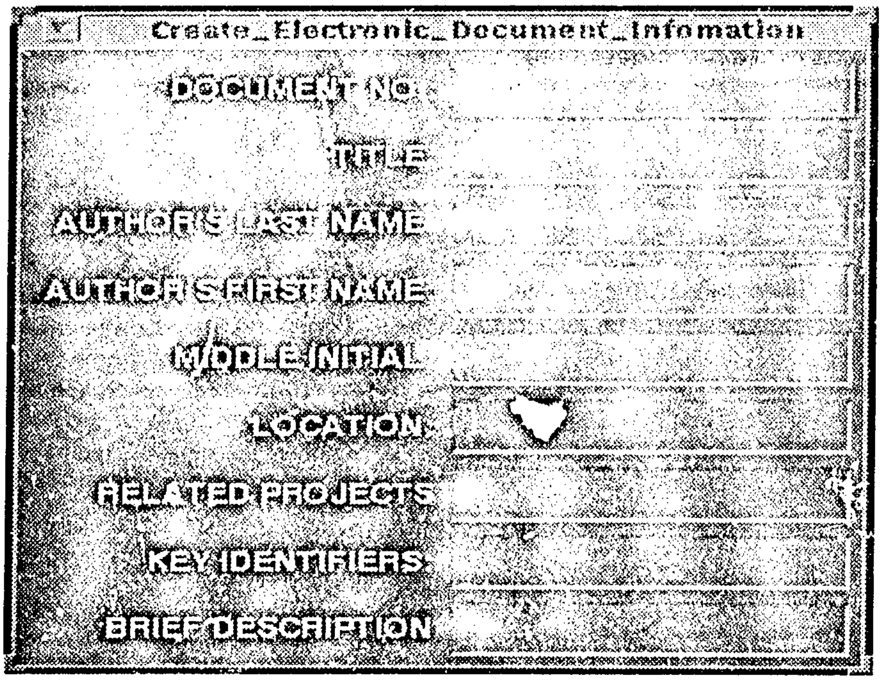


Figure H.5. Creating Electronic Document Information Popup

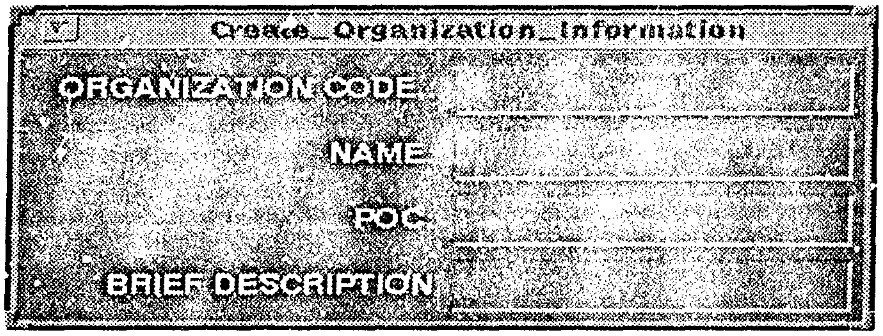
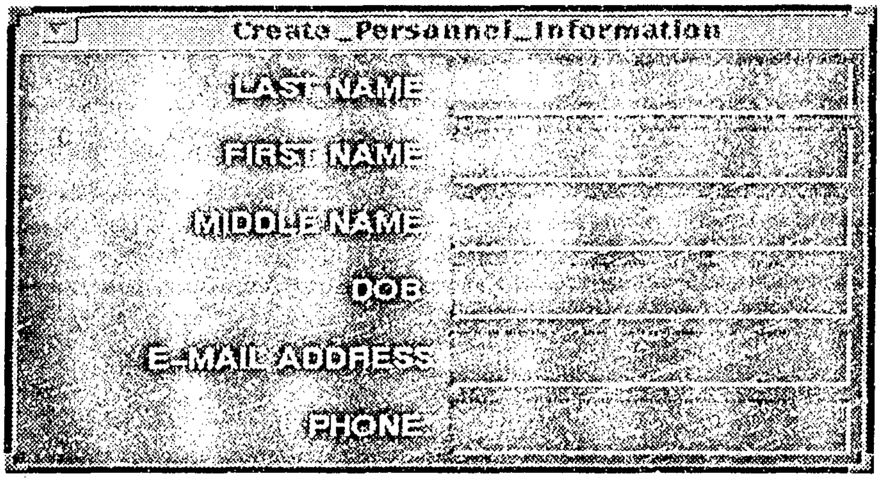


Figure H.6. Creating Organization Information Popup



Create_Personnel_Information

LAST NAME

FIRST NAME

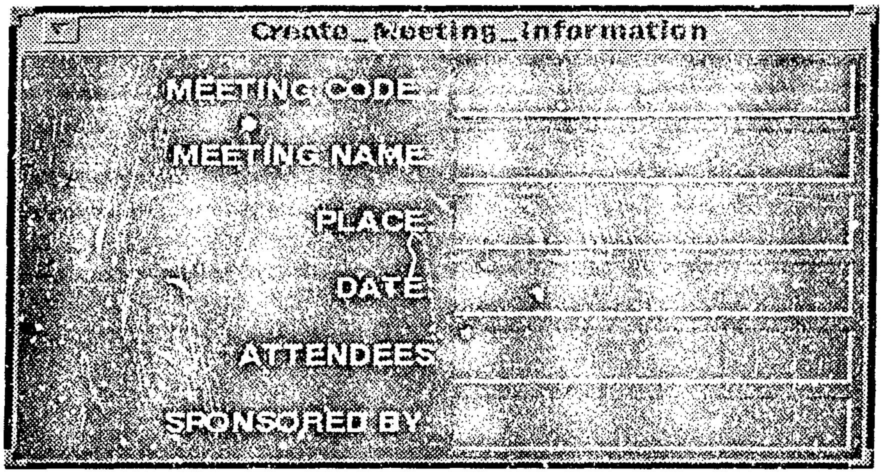
MIDDLE NAME

DOB

E-MAIL ADDRESS

PHONE

Figure H.7. Creating Personnel Information Popup



MEETING CODE

MEETING NAME

PLACE

DATE

ATTENDEES

SPONSORED BY

Figure H.8. Creating Meeting Information Popup

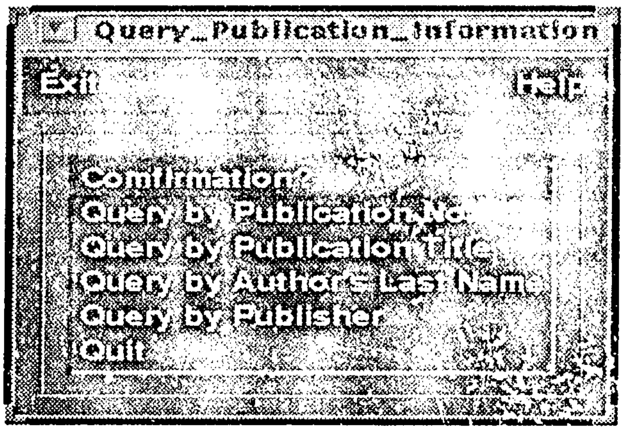

Query_Publication_Information
Exi
Help
Comfirmation?
Query by Publication No
Query by Publication Title
Query by Authors Last Name
Query by Publisher
Quit

Figure H.9. Querying Publication Information Popup

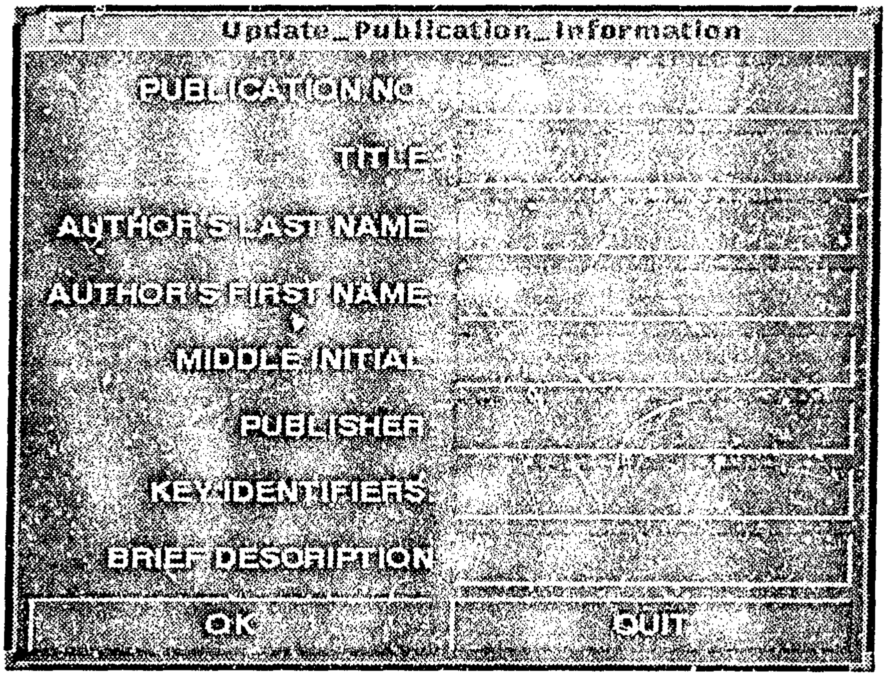


Figure H.10. Updating Publication Information Popup

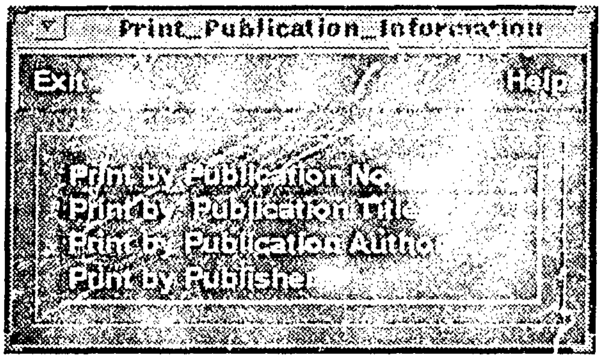
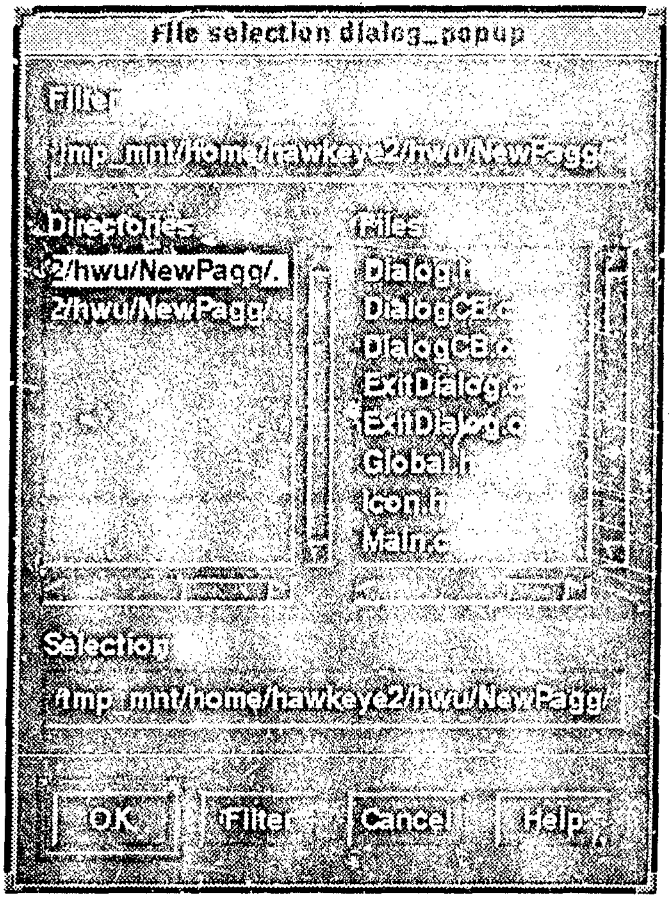

Print_Publication_Information
Exit
Help
Print by Publication No.
Print by Publication Title
Print by Publication Author
Print by Publisher

Figure H.11. Printing Publication Information Popup



/tmp_mnt/home/hawkeye2/hwu/NewPagg/

2/hwu/NewPagg/.
2/hwu/NewPagg/..

Dialog.h
DialogCB.c
DialogCB.o
ExitDialog.c
ExitDialog.o
Global.h
Icon.h
Main.c

/tmp_mnt/home/hawkeye2/hwu/NewPagg/

Figure H.12. DOODBMS Application User Interface— File Selection Panel

Bibliography

1. Aaron Marcus, Andries Van Dam, "User-Interface Developments for the Nineties" Computer, IEEE Vol 24 , September, 1991.

2. "Ada and C++: a business Case Analysis", Deputy Assistant Secretary of the Air Force(Communications, Computers, and Logistics), Washington, D. C. 20330-1000. 1991.

3. "Ada and C++: A Lifecycle Cost Analysis.", TRW Inc. Space and Defense Sector, Contract No. MDA970-89-C-0019(prepared for SAFAQKS June 1, 1991. in "Ada and C++: a business Case Analysis", Deputy Assistant Secretary of the Air Force(Communications, Computers, and Logistics), Washington, D. C. 20330-1000.

4. Ahad, Rafiui ; Dedo, Douglas. " OpenODB from Hewlett-Packard: a commercial object-oriented database management system " Journal of Object-Oriented Programming, SIGS Publication, Ins., volume 4, number 9, page 31, Feb. 1992.

5. Shamim Ahmed, Labert Wong, Duvvuru Sriram , Robert Logcher , " A Comparison of Object-Oriented Database Management Systems for Engineering Applications " Massachusetts Institute of Technology, Engineering Systems Lab and Department of Civil Engineering. Research Report R91-12, Order No.IESL 90-03,91-03. 1991.

6. Shamim Ahmed, Labert Wong, Duvvuru Sriram , Robert Logcher , "Object-Oriented Database Management Systems for Engineering Applications: a Comparison " Journal of Object-Oriented Programming, SIGS Publication, Ins., volume 5, number 3, page 27, June 1992.

7. Alagic, Suad . "Object-oriented Database Programming" Springer-Verlag, New York Inc. 1989.

8. Andleigh, Prabhat K., Gretzinger, Michael R., "Distributed Object-Oriented Data-Systems Design", PTR Prentice Hall, Englewood Cliffs, New Jersey 07632, 1992.

9. Andrews, T., Andrews, C., "Combining Language and Database Advances in an Object-oriented Development Environment.", in Proceeding second International Conference Object-oriented Programming System, Languages, Applications, Orlando, FL. Oct. 1987, pp. 430-440.

10. Atkinson et. al, "The Object-Oriented Database System Manifesto" in "Deductive and Object-Oriented Databases", Elsevere Science Publishers, Amsterdam, Netherlands, 1990,

11. Bancilhon, Francois; Kim, Won; " Object-oriented Database Systems: in Transition", SIGMOD Record (ACM Special Interest Group on Management of Data) vol.19 No. 4 Dec 1990 p 49-53, 1990.

12. Bancilhon, Francois. "A classification of Object-oriented Database Systems" in " Database Programming Languages: Bulk Types and Persistent Data", Kanellakis, Paris and Joachim W. Schmidt, Morgan Kaufmann Publishers, San Mateo, Cal. 1992.

13. J. Banerjee, W. Kim, H. F. Korth, "Semantics and Implementation of Schema Evolution" in "Object-oriented Database System." in proceedings of the ACM SIGMOD Conference, 1987.

14. David Bell, Jane Grimson, "Distributed Database Systems" Addison-Wesley Publishers Ltd. 1992.

15. Bertino, Elisa; Negri, Mauro and Licia Sbattella, "Object-oriented Query Languages: the Notion and the Issues.", IEEE Transaction on Knowledge and Data Engineering, Vol. 4, No. 3, June 1992.

16. Bertino, E. , L.M. Hass , and B.G. Lindsay , "View Management in Distributed Database System" , Proc. 1983, VLDB, pp. 376-378, Oct. 1983.

17. Blaha, Michael, "Models of Models", Journal of Object-Oriented Programming, September, 1992, vol. 5, No. 5, pp. 13-18.

18. Blair, Gordon S.; Lea, Rodger; "The Impact of Distribution on Support for Object-Oriented Software Development", Software Engineering Journal, March 1992.

19. Boehm, B. W., "Software Engineering Economics", Englewood Cliffs, NJ:Prentice-Hall, 1981.

20. Boehm, B. W., "A Spiral Model of Software Development and Enhancement.", IEEE Computer, Vol. 21, No. 5, May 1988.

21. Boehm, B. W., "Software Risk Management: Principles and Practices.", IEEE Software, Vol. 8, No. 1, January 1991.

22. Bollay, Denison, "Dylan(dynamic language) or CLOS +-", Journal of Object-Oriented Programming, November-December 1992.

23. Booch, Grady , "Object-oriented Design with Application", Bejemming Publishing Co. , 1991.

24. Bray, H. Olin , "Distributed Database Management ", Lexington Book Publishing company , Massachusetts,1982.

25. Yuri Breitbart, "Database Integration In A Distributed Heterogeneous Database System", IEEE CS Press, Los Alamitos, Calif., 1986, pp. 301-310.

26. Brooks, F. P. Jr., "No Silver Bullet: Essence and Accidents of Software Engineering.", Computer vol. 24, No. 4 April 1987, pp. 10-19.

27. Carey, M., et al. "The Architecture of the EXODUS Extensible DBMS" in Proceeding of the 1986 International Workshop on Object-Oriented Database Systems, ACM, SIGMOD, New York, 1986.

28. Cattell, R.G.G., "Object Data Management ", Addison-Wesley Publishing company 1991.

29. Cattell, R.G.G.; Skeen , J ; "Object Operations Benchmark", ACM Transactions on Database Systems, March 1992 Volume 17 number 1.

30. Ceri, Stefano , "Distributed Database", McGraw-Hill Book Company 1984. New York.

31. Peter P. Chen, " The entity-relationship model: Toward a unified view of data." ACM Transactions on Database Systems, 1(1), 1976.

32. Chu, Wesley W. , "Optimal File Allocation in a Multiple Computer System ", IEEE Trans. vol. c-18 no. 10, pp.885-889,1969.

33. Coleman, Derek and others, "Introducing Objectcharts or How to Use Statechart in Object-Oriented Design", IEEE Transaction on Software Engineering, vol. 18, no. 1, 1992.

34. Coad, P. and Yourdon, E. , "Object-oriented Analysis", Englewood Cliffs, NJ: Prentice-Hall, 1991.

35. Coad, P. and Yourdon, E. , "Object-oriented Design", Englewood Cliffs, NJ: Prentice-Hall, 1991.

36. Coad, E. F, "Database Programming and Design", Volume 4, NO. 3 pp. 49, Arch 1991.

37. Computer Select(Stand alone), Apr. 1992 Data Source Report, Ziff-Davis Publishing Co. 1992.

38. Date, C. J. "An Introduction to Database Systems" fifth edition, Addison Wesley Publishing Co. , Inc. 1991.

39. Davis, A. "Software Requirements: Analysis and Specification", Prentice Hall, Englewood Cliffs, N.J., 1990.

40. Davis, A. "Operational prototyping: the POST story submitted to IEEE Soft", 1991.

41. de Champeaux, Dennis and Penlope Faure, "A Comparative Study of O-O Analysis Methods", Journal of Object-Oriented Programming, March/April, 1992, Vol. 5, No. 1, pp. 21-33.

42. de Champeaux, Dennis; Personal conversations(through E-mail) , January 7, 1993, (ddc@hplddc.hpl.hp.com) Research scientist in HP.

43. Dowdy , L.W. and Foster D.V. "A Comparative Models of the File Assignment Problem ", Computer Surveys , vol 14, 1982 ,pp. 287-313.

44. Dyer, Douglas E. and Mark A. Roth. "Object-Oriented Design Unifies Databases and Applications", Tech. Rep., AFIT/EN-TR-92-2, Air Force Institute of Technology, AFIT/LD Bldg. 640 Area B Wright-Patterson AFB, OH, 45433, July 1992.

45. Edelstein, Herbert. "Database World Targets Next-generation Problems", Software Magazine, May 1991. pp. 78-86.

46. Edelstein, Herb. "Distributed DBMS", Journal: Computerworld Nov 4 1991 v25 n44 p77(5).

47. Hector Garcia-Molina and Bruce Lindsay , "Research Directions for Distributed Databases ", SIGMOD RECORD, vol. 1, pp.98-103 , 1990.

48. Servio Logic Development Inc. (Product introduction), " Code-free development offered for OODBMS" Journal: Software Magazine , v12 n3 p87(1) March 1992.

49. V.D. Gligor and G.L. Luckenbaugh, "Interconnecting Heterogeneous Database Management Systems", IEEE Computer, January 1984.

50. Gray, P. M. D.; Kularni, K. G.; Paton, N. W.; "Object-Oiiented Databases—a Semantic Data Model Approach", Prentice Hall, Englewood Cliffs, NJ 07632, 1991.

51. Gray, Mark H., de Barr, Dennis J. M. J. James D. Foley, KevinMullet; "Coupling Application Design and User interface Design", Human Factors in Computing Systems, CHI'92 Conference proceedings. ACM Press 1992.

52. Green, Mark, " The University of Alberta User Interface Management System", in Procedings of SIGGRAPH '85, 12th Annual Conference (San Francisco, Calif., July 22-26). ACM, New York, pp. 205-213.

53. Dawn Guido, Class handouts, AFIT/ENG faculty , Air Force Institute of Technology, AFIT Bldg. 125 Area B Wright-Patterson AFB, OH, 45433-6583, 1992.

54. Dawn Guido, Conversation of appointment at 10:10 Oct. 23 1992, AFIT/ENG faculty , Air Force Institute of Technology, AFIT Bldg. 125 Area B Wright-Patterson AFB, OH, 45433-6583.

55. Rajiv Gupta , Ellis Horowitz , Editors, "Object-Oriented Database with Applications to CASE, Networks, and VLSI CAD", Prentice Hall, Englewood Cliffs, NJ 07632, 1991.

56. "Object-Oriented database Management System Products", Object-Oriented Strategies, Editor: Paul Harmon, Cutter Information Corp. Vol. 2, No. 2 , 1992.

57. Hartson, H. Rex and Deborah Hix, "Human-Computer Interface Development: Concepts and Systems for Its Management", ACM Computing Survey, March 1989, pp. 5-92.

58. Hartrum, Thomas C., "Easy Guide to OOD(Based on Rumbaugh et al.)", Air Force Institute of Technology, AFIT/LD Bldg. 640 Area B Wright-Patterson AFB, OH, 45433-6583, 1992.

59. Harwood, O. E. , "Analysis and Development of a Distributed Database Design Methodology for the United States Army Manumeuver Control System", Master Thesis, AFIT/GCS/ENG/86J-3, Air Force Institute of Technology, AFIT/LD Bldg. 640 Area B Wright-Patterson AFB, OH, 45433-6583, 1986.

60. Hayes, Fiona ; Coleman, Derek, "Coherent Models for Object-oriented Analysis" ACM OOPSLA '91 Conference Procedings, 1991.

61. Henderson-Sellers, Brian and Julian M. Edwards. "Object-oriented Systems Lifecycle" *Communications of the ACM*, pp. 142-159, Sep. 1990

62. Hewlett-Packard Company, HP Interface Architecture Developer's Guide,Hewlett-Packard Company, Corvallis, Oregon, October 1990.

63. Kunihiko Higa and Olivia R. Liu Sheng. "An object-oriented methodology for end-user logical database design: The structured entity model approach" In *Proceedings of the Thirteenth Annual International Computer Software and Applications Conference.* IEEE Computer Society, 1989.

64. Hughes, J. G.; "Object-Oriented Databases" Prentice Hall, Englewood Cliffs, NJ, 07632, 1991.

65. Itasca System Inc. "Itasca Technical Summary Release 2.0", ITASCA Systems, Inc. Sep. 1991.

66. Itasca System Inc. "Itasca C++ API User Manual(Part 1) Release 2.1", ITASCA Systems, Inc. 1992.

67. Itasca System Inc. "Itasca C++ API User Manual(Part 2) Release 2.1", ITASCA Systems, Inc. 1992.

68. Itasca System Inc. "Itasca Dynamic Schema Editor User Manual Rele·.se 2.1", ITASCA Systems, Inc. 1992.

69. Itasca System Inc. "Itasca DBATool User Manual Release 2.1", ITASCA Systems, Inc. 1992.

70. Jablonski, S.; Ruf, T.; Wedekind, H.; "Implementation of a distributed data management system for technical applications. A feasibility study", Information Systems, vol. 15 no. 2 1990, pp. 247-256. 1990.

71. Jazayeri, Mehdi, "Distributed Software Design Techniques", in "Advances in Object-Oriented Software Engineering "(Chapter 6), edited by Dino Mandrioli and Bertrand Meyer, Prentice Hall, 1992 pp. 147-165.

72. Jhingran, Anant and Michael Stonebraker. "Alternatives in Complex Object Representation: A Performance Perspective." In *Proceedings of the Sixth International Conference on Data Engineering*, pages 94–102, February 1990.

73. Johnston, Marsha W. " French OODBMS newcomer strong at starting gate; O2 Technology seeks differentiation to extend popularity beyond France.", Journal: Software Magazine , v12 n3 p77(3), March 1992.

74. Johnson, Eric F. and Kevin Reichard, "Power Programming ", MIS Press, Portland, 1991.

75. Jones, Oliver , "Introduction to the X Window System", Prentice-Hall Inc. 1989.

76. Joseph, John V.; Thatte, Satish M.; Thompson, Craig W.; Wells, David L. "Object-oriented databases: Design and implementation", Proceedings of the IEEE Vol. 79, No. 1 January, 1991 p 42-64, 1991.

77. Kemerer, Chris. "How the Learning Curve Affects CASE Tool Adoption", IEEE Software, May, 1992.

78. Ketabchi, M. A., Mathur, S., Risch, T., and Chen, J., "Comparative Analysis of RDBMS and OODBMS: A Case Study", Proceeding of Compcon IEFE Computer Society International Conference, San Fransisco, CA. Feb. 1990

79. Khoshafian, Setrag and Razmik Abnous, "Object-Orientation: Concepts, Languages, Databases, User Interfaces", Wiley and Sons, 1990, New York.

80. Won Kim, Nat Ballou, Jay Banerjee, Hong-Tai Chou, Jorge F. garza, darrel Woelk, "Features of the ORION Object-Oriented Database System." The proceedings of the 13th International VLDB Conference, 1987, pp. 319-329. Object-oriented Databases. IEEE Computer Society Press.

81. Kim, Won et al., "Composite object support in an Object-oriented Database System," in Proceeding second International Conference Object-oriented Programming System, languages, Applications, Orlando, FL. October 1987.

82. Kim, Won et al., "Integrating an Object-oriented-oriented Programming System with a Database System." in Proceeding second International Conference Object-oriented Programming System, Languages, Applications, San Diego, CA. , Sep. 1988.

83. Kim, Won and others. "Indexing Techniques for Object-Oriented Databases." In *Object-Oriented Concepts, Databases, and Applications*, chapter 15, pages 371–394, New York: ACM Press, 1989.

84. Kim, Won; "Object-oriented databases: Definition and research directions", IEEE Transactions on Knowledge and Data Engineering, vol. 2 No. 3 Sep 1990. pp. 327-341.

85. Kim, Hyoung-Joo. "Algorithmic and Computational Aspects of OODB Schema Design." in " Object-Oriented Databases with Application to CASE, Networks, and VLSI CAD", Gupta, Rajiv and Ellis Horowitz, Eds. Englewood Cliffs, New Jersey: Prentice-Hall, 1991, chapter 3, page 26-61.

86. Kosyachenko, S. A.; Kul'ba, V. V.; Mamikonov, A. G.; Uzhastov, I. A. " Distributed data base design: models and methods.", Automation and Remote Control (English translation of Avtomatika i Telemekhanika) 50 7 PT 1, p 855-897, Dec. 1989.

87. Korson, Tim and Johnu U. McGregor, "Understanding object-oriented: a unifying paradigm", Communications of the ACM", September", 1990", vol. 33, pp. 41-60.

88. Korzeniowski, Paul. " Object-oriented DBMSs strive to differentiate", Journal: Software Magazine ,v12 n6 p68(5) May 1992.

89. Korth, Henry F., "Database System Concept", second edition, McGraw-Hill Publishing company, New York, 1991

90. Krasowski, Michael; "Why Choose Distributed Database?" Database programming and Design, volume 4, number 3, March 1991, Miller Freeman Publications, CA.

91. Gerti Kappel and Michael Schreff. "A behavior integrated entity-relationship approach for the design of object-oriented databases" In C. Batini, editor, *Entity-Relationship Approach.* Elsevier Science Publishers B.V. (North-Holland), 1989.

92. Lamb, Charles., Landis, Gordon. , Orenstein, Jack. and Weinreb, Dan, "The ObjectStore Database System", Communications of the ACM, Vol. 34, No. 10, pp.51-63, 1991.

93. Lamont, Gary B., Hsin-feng Wu; "Comments on the Object-Oriented Approach to Software Engineering" in support of CSCE586 – Advanced Information Structures for Software Engineering in the Small, Fall Quarter 1992. AFIT, WPAFB, Ohio, 45433.

94. Larson , A. James and Rahimi, Saeed , "Decomposition Requests", IEEE DDBM, Dec. 1985 pp. 91-94.

95. Lawlis, Patricia K. and Drew Hamilton, "ASEET Tutorial: Object-Oriented Concepts", AFIT and Army Computer Science School January 1993.

96. Lee, Ed, "User-Interface Development Tools.", IEEE Software , May 1990.

97. Lim, Samkyu, "Transition to Ada 9X", CSCE699 Independent Study, Supporting faculty: Col. Lawlis, Patricia K., Winter Quarter 1993, AFIT, WPAFB, Ohio, 45433.

98. Lohman, G. M., C. Mohan, and L. Hass, D. J. Daniels, B. Lindsay, P. Selinger, and P. Wilms. "Query Processing in R*".

99. Mary E. S. Loomis, "Building Object database applications", Journal of Object-Oriented Programming, SIGS Publication, Ins. June 1992.

100. Lorensen, W., "Object-Oriented Design, CRD Software Engineering Guidelines", General Electric Co., 1986.

101. Lyngbaek, Peter; Dennis McLeod, "Object management in Distributed Information Systems" in "Research Foundations in Object-Oriented and Semantic Database Systems" Edited by: Cardenas, Alfonso F., Dennis McLeod, Prentice-Hall, 1990.

102. Maier, David, Jacob Stein, Alan Purdy, "Development of an Object-Oriented DBMS" in "Research Foundations in Object-Oriented and Semantic Database Systems" Edited by: Cardenas, Alfonso F., Dennis McLeod, Prentice-Hall, 1990.

103. Martin, James and James J. Odell. "Object-Oriented Analysis and Design." Englewood Cliffs, New Jersey: Prentice-Hall, 1992.

104. McCormick, W. T., Schweitzer, P. J. and White, T. W. "Problem decomposition and data recognition by a clustering technique", Operation Research 20, 5 Sep. 1972, pp. 993-1009

105. McMinds, Don , Research scientist in HEWLETT PACKARD User Interface Technology Division. Personal conversation through E-mail during December 1992 through Febuary 1993. (email address: dlm@hpcvlx.cv.hp.com Telnet 750-3244)

106. Mohan, C., "Tutorial: Recent Advance in Distributed Data Base Management", Computer Society Press , DDBM/T/IEEE, 1984.

107. Monarchi, David E. and Gretecen I. Puhr, "A Research Typology for Object-Oriented Analysis and Design" Communication of the ACM, Sep. 1992, Vol. 35, No. 9.

108. Mosley, Vickey. "How to Access Tool Efficiently and Quantitatively", IEEE Software, May 1992.

109. Mukkamala, Ravi, " Measuring the effects of data distribution models on performance evaluation of distributed database systems.", IEEE Transactions on Knowledge and Data Engineering vol. 1 no. 4. pp. 494-507, Dec. 1989.

110. Myers, Brad A., "User-interface Tools: Introduction and Survey", (a research sponsored by the Defense Dept.'s Advanced Research Projects Agency under order 4976, under contract F33615-87-C-1499and monitored by the Avionics Lab, Air Force Wright Aeronautical Lab, Aeronautical Systems Div. WPAFB, OH) IEEE Software, January 1989.

111. Meyer, Bertrand. "Object-Oriented Software Construction." Prentice Hall, 1988.

112. Navathe , Shamkant,B. and Ceri,Stenfano , " A Comprehensive Approach to Fragmentation and Allocation of Data in Distributed Database", IEEE DDBMS/T 1985 , pp. 122-134.

113. Navathe , S. , Ceri,Stenfano and Wiederhold Dou, "Vertical Partitioning Algorithms for Database Design", ACM Trans. Vol. 9 no. 4 pp.680-710, Dec 1984.

114. Navathe , S. , Ceri,Stenfano and Wiederhold Dou, "Vertical Partitioning Algorithms for Database Design", ACM Trans. Vol. 9 no. 4 pp.680-710, Dec 1984.

115. Nerson, Jean-Marc; "Applying Object-Oriented Analysis and Design" Communication of the ACM, vol. 35, no. 9, September 1992.

116. NeXt Computer Inc., " NeXTstep Concepts", Redwood City, CA.: NeXTComputer, Inc., pp. 8-1 to 8-35.

117. Norman,M., and M. Anderson ,"EMPACT:A Distributed Database Application", AFIPS Conference Proc. Vol. 52 , pp.203-217, 1983.

118. Shamkant B. Navathe and MohanK. Pillalamarri. "OOER: Toward making the E-R approach object-oriented" In C. Batini, editor, *Entity-Relationship Approach.* Elsevier Science Publishers B.V. (North-Holland), 1989.

119. Oman, Paul, "CASE Analysis and Design Tools.", IEEE Software , May 1990.

120. Open Software Foundation. " OSF/Motif Style Guide", Revision 1.0, OSF Cambridge Center, Cambridge, MA 02142, 1990.

121. Ozsu, M Tamer, "Principles of Distributed Database Systems",1991 Prentice Hall Englewood Cliffs, New Jersey

122. Ozsu, M. Tamer ; Patrick Valduriez, "Distributed Database Systems: Where Are We Now?" Computer, IEEE Vol. 24 August, 1991.

123. Ozsu, M. Tamer; Personal conversations(through E-mail : ozsu@cs.ualberta.ca) Associate professor, Faculty member of in the Department of Computing Science st the University of Alberte (Edmonton, Canads), Nov. 23, 1992.

124. Parker, Charles, Technical Support representitive, CenterLine Incorporate. Technical Support through E-mail during January 1992 through Febuary 1993. (email address: cparker@centerline.com)

125. Pascoe, Geoffrey A., "Elements of Object-oriented Programming" BYTE 1986. in Peterson, Gerald E., (editor) "Tutorial: Object-Oriented Computing. Volume 1: Concepts" IEEE, 1987.

126. Pausch, Randy and others. "Lessons Learned from SUIT, the Simple User Interface Toolkit" ACM Transactions on Information Systems, Vol. 10, No. 4, Oct. 1992, pp. 320-344.

127. Peterson, Gerald E., "Tutorial: Object-Oriented Computing. Volume 2: Implementations" IEEE, 1987.

128. Peterson, Gerald E., "Tutorial: Object-Oriented Computing. Volume 1: Concepts" IEEE, 1987.

129. "The Muse of objects. (Persistent Objects and Extended database Technology, or POET, low-cost C++ object-oriented database management system from German-based BKS) (Product Announcement)", Journal: EXE, v6 n8 p4(1), Feb 1992.

130. Polese, K. and T. Goldstein; "OOP Languages and Methodologies", SunWorld, May 1991 , pp. 45-59

131. Poston, Robert, Michale P. Sexton. "Evaluating and Selecting Testing Tools", IEEE Software, May 1992.

132. Roger S. Pressman. "Software Engineering: A Practitioner's Approach (Third Edition).", New York: McGraw-Hill Publishing Company, 1991.

133. Premeriani, William J.; Blaha, Michael R.; Rumbaugh, James E.; Varwig, Thomas A. "Object-oriented relational database", Communications of the ACM v 33 n 11 Nov 1990 p 99-109, 1990.

134. Pages-Jones, M. , "The practical guide to Structure System Design", Englewood Cliffs, NJ: Yourdon Press, pp.261-265.

135. "OSF/Motif Programmer's Guide". January 1992. Quest Windows Coporation, 5200 Great America Parkway, Santa Clara, CA. 95054.

136. "OSF/Motif Style Guide", January 1992. Quest Windows Coporation, 5200 Great America Parkway, Santa Clara, CA. 95054.

137. Radding P. L., "Achieving High-Quality Systems: Making CASE Works in Your Organization.", IEEE, 1990.

138. Ramamoorthy, C. V. ,Sheu, P. C., and Chillakanti, P. "Object-oriented Approach: System, Programming, Language, and Applications", IEEE , Object-oriented Database, 1991.

139. Sudha Ram, "Heterogeneous Distributed Database Systems", Computer, Vol.24, December, 1991, pp.7-8.

140. Naphtali Rishe, "Database Design Fundamentals", Prentice Hall, Englewood Cliffs, New Jersey, 1988.

141. Roth, Mark A., Personal conversations, Faculty member of Air Force Institute of Technology,(School of Engineering), March 1992 through March 1993.

142. Rofrano Jr. , J. J. , "Design Considerations for distributed applications" IBM System Journal Vol 31 No 3 1992.

143. Routhier, Kevin J. , " Entity-Relationship Versus Object-Oriented Modeling and the Underlying DBMS" AFIT/GCS/ENG/92D-15, Master thesis, Air Force Institute of Technology, AFIT/LD Bldg. 640 Area B Wright-Patterson AFB, OH, 45433-6583, Dec. 1992

144. Rumbaugh, James , Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, "Object-oriented Modeling and Design", Englewood Cliffs, NJ: Prentice Hall, 1991.

145. James Rumbaugh, "Modeling and Design: A National Identity Crisis", Journal of Object-Oriented Programming, page 11, SIGS Publication, Ins. October 1992.

146. Sacca,D., and G. Wiederhold, "Database Partitioning in a Cluster of Processors", ACM Transactions on Database Systems, vol. 10,No.1,pp.29-56,March 1985.

147. Schooler, Tony. Custumer Support(Comm. Computer Center) Room 2200, Air Force Institute of Technology, AFIT/LD Bldg. 640 Area B Wright-Patterson AFB, OH, 45433-6583, 1993.

148. "The ISO/ANSI SQL standard: where are we? (International Standards Organization, The American National Standards Institute; structured query language)" Journal: Patricia Seybold's Unix in the Office , v6 n11 p6(2), Nov 1991.

149. Sheth, Amit; James A. Larson; "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Database" ACM Computing Surveys, Vol. No. 3, 1990.

150. Shlaer, S., and Mellor, S. J., "Object-Oriented systems analysis", Yourdon Press, Prentice Hall, 1988.

151. Shneiderman, B., "Designing the User Interface: Strategies for Effective Human-Computer Interaction", Addison-Wesley, Reading, MA., 1987.

152. Sikkel, K. ; Van Vliet; "Abstract Data Types as Reusable Software Components: the Case for Twin ADTs", Software Engineering Journal, May 1992.

153. Avi Silberschatz, Michael Stonebraker and Jeffrey Ullman "Database Systems: Achievements and Opportunities", December 1990 ACM Sigmod Record.

154. Simpson, Dennis Joseph , "An Application of the Object-Oriented Paradigm to a Flight Simulator", AFIT/GCS/ENG/91D-22, Master thesis, Air Force Institute of Technology, AFIT/LD Bldg. 640 Area B Wright-Patterson AFB, OH, 45433-6583, 1991.

155. Smith, Dennis B., Paul W. Oman; "Software Tools in Context", IEEE Software, May 1990.

156. Spragins, John D., Joseph L. Mammond and Krzysztof Pawlikowski; "Telecommunications: Protocols and Design", Addison-Wesley Publishing Company, 1991. pp.120-122.

157. Stevens, W. Richard, "Unix Network Programming", PTR Prentice-Hall Inc. 1990.

158. Stonebraker, M. and Rowe, L. "The Design of POSTGRES". in Proceedings of SIGMOD '86 International Conference on Management of Data, ACM, SIGMOD, New York, 1986.

159. Stonebraker, Michael; Lawrence A. Rowe; Bruce Linasay, James Gray,Michael Carey, Michael Bordie, Philip Bernstein, and David Beech. "Third-generation Database System Manifesto", SIGMOD RECORD Vol. 19, No. 3, September 1990. (in proceedings of the IFIP DS-4 Workshop on Object-Oriented Databases, Windermere, England, July 1990.)

160. Stroustrup, Bjarne. "What is Object-Oriented Programming?", IEEE Software, Vol. 5, pp. 10-20, May 1988.

161. Stroustrup, Bjarne. "The C++ Programming Language", Second Edition, Addison Wesley Publishing Co. 1991.

162. Sun Microsystems, Inc. and AT&T, "OPEN LOOK GUI Application Style Guidelines, Addison", Wisley, Reading, MA, 1990.

163. Sun Microsystems, Inc. "Open Windows Developer's Guide 1.1", Reference Manual, June 1990.

164. Sutcliffe, A. G. "Object-oriented systems development. Survey of structured methods", Information and Software Technology v 33 n 6 Jul-Aug 1991 p 433-442, 1991.

165. Ullman, Jeffrey,D., "Database and Knowledge-Base System", vol. 1, Computer science press company , Maryland , 1988.

166. Urban, Susan D., Chalmers, Kimberly, "An Investigation of the View Update Problem for Object-Oriented Views", IEEE, 1992.

167. Itasca System Inc. "Itasca User Manual Release 1.0", Sep. 1991.

168. Wah,B.W. , "File Placement on Distributed Computer System", Computers, vol. 17, no. 1 pp. 23-32

169. Walker, Ian J., "Requirements of an object-oriented design method" Software Engineering Journal, March 1992.

170. Paul T. Ward. "How to integrate object orientation with structured analysis and design", *IEEE Software*, March 1989.

171. Wasserman, A. J., and Muller, R. J., "The Object-Oriented Structure Design Notation for Software design representation", Computer , IEEE, 1990, pp. 50-63.

172. Wiederhold, G. "Database Design"(2nd edition). New York: McGraw-Hill, 1982.

173. Wileden, Jack C.; Alexander L. Wolf; William R. Rosenblatt; Peri L. Tarr; "Interoperability", Communications of the ACM, Vol. 34 No. 5, May 1991.

174. Willshire, Mary Jane, "How Spacey Can They Get? Space Overhead for storage and Indexing with Object-Oriented Databases" IEEE 7th International Conference On Data Engineering, 1991.

175. Wirfs-Brock et al. "Designing Object-Orient^d Software". Prentice Hall, Englewood Cliffs, N. J., 1990.

176. Wolf, Wayne; "A Practical Comparison of Two Object-Oriented Languages", IEEE Software, September 1989.

177. Wessel, Chuck; Technical Support representitive, ITASCA company. Technical Support through E-mail during October 1992 through Febuary 1993. (email address: support@itasca.com)

178. M.F. Worboys and S.M. Deen, "Semantic Heterogeneity in Distributed Geographic Datab ses", SIGMOD RECORD, Vol.20, Dec. 1991.

179. Anthony I. Wasserman, Peter A. Pircher, and Robert J. Muller. The object-oriented structured design notation for software design representation. *Computer*, March 1990.

180. xnetlib(3.3) - an X interface to netlib. The University of Tennessee and Oak Ridge National Laboratory. Dec. 1992.

181. Young, Douglas, "The X Window System: Programming and Applications with Xt (OSF/Motif Edition)", Prentice Hall, Englewood Cliffs NJ, 1990

182. Yourdon, Edward, "Modern Structured Analysis", Englewood Cliffs, NJ: Yourdon Press, 1989.

Vita

Captain Hsin-feng Wu was born on July 2, 1965, in Koahsiung, Taiwan, Republic of China (ROC). He graduated from the National Military Preparatory High School in Koahsiung, a southern city in Taiwan, in 1983. He entered the ROC Air Force Academy in 1983, where he received the Bachelor of Science degree in 1987. Upon graduation he was assigned as a first lieutenant of the Air Force. He entered the School of Engineering, Air Force Institute of Technology of United States, in June, 1991.

Permanent address: 9 Lane 283 Mei-shan Rd.
Kaohsiung, Taiwan, R.O.C. 83301

March 1993 Master's Thesis

A DISTRIBUTED OBJECT-ORIENTED DATABASE APPLICATION DESIGN

Hsin-feng Wu

Air Force Institute of Technology (AFIT), WPAFB OH 45433-6583

AFIT/GCS/ENG/93M-06

Robert F. Pollicik
WL/DOIT WPAFB, DAYTON, OH, 45433

Distribution Unlimited

Abstract

The purpose of this study is to analyze and develop a distributed object-oriented database management system (DOODBMS) application to support parallel software development of the Parallel Algorithms and Applications Group (PAAG) at AFIT. By following the software lifecycle of object-oriented paradigm, this thesis investigation is intended to generate requirements analysis, design, implementation, user interface design and implementation, and integration of the database application with the user interface. ITASCA, an existing DOODBMS platform, has been chosen to prototype the application for its distribution functionalities and object-oriented modeling power. The user interface of the database application is based on X-window OSF/Motif front-end system for its efficiency. Various operations of this distributed database include: retrieve information in the database, modify data at the local site, look up information of publications of libraries, etc. The user may access each local computer through graphical user interface without knowing the query language constructs of the DOODBMS platform.

Database management system (DBMS), Object-oriented database management system (OODBMS), Distributed object-orienteddatabase management system (DOODBMS), User interface application design, Motif user interface design

214

UNCLASSIFIED UNCLASSIFIED UNCLASSIFIED UL